LINK DISCOVERY: ALGORITHMS AND APPLICATIONS

Der Fakultät für Mathematik und Informatik
der Universität Leipzig
eingereichte

HABILITATIONSSCHRIFT

zur Erlangung des akademischen Grades

DOCTOR RERUM NATURALIUM HABILITATUS
(Dr. rer. nat. habil.)

im Fachgebiet Informatik
vorgelegt
von

Dr. rer. nat. Axel-Cyrille Ngonga Ngomo

geboren am 04.08.1983 in Bafoussam, Kamerun

Axel-Cyrille Ngonga Ngomo: *Link Discovery: Algorithms and Applications*

To my loving families. Thank you for your love and support.

ABSTRACT

The current transition from the Document Web to the Semantic Web has led to the current precursor of the latter, i.e., the Linked Data Web. While the Linked Data Web began as a set of only 12 knowledge bases, it has grown steadily since and is now a compendium of close to 10,000 knowledge bases that contain approximately 150 billion facts.[1] Setting links between the entities described in these knowledge bases is of central importance to implement a large number of the applications such as cross-ontology question answering, large-scale inference and data fusion.

Two main challenges arise when aiming to link between knowledge bases: First, the mere size of knowledge bases would result in impractical runtimes if links were set manually. We call this challenge the *time-complexity challenge*. In this work, we consider declarative solutions to this challenge, where combinations of bounded similarity or distance functions are used to compute an approximation of the set of links between knowledge bases. Within this setting, using naïve solutions for computing links also leads to impractical runtimes due to the quadratic complexity of the link discovery problem. The provision of scalable declarative solutions for link discovery is hence the first challenge that we address.

The second challenge that arises within the declarative setting of link discovery is the *accuracy challenge*. Scalable solutions are of little use if their results display poor precision or recall. Finding the right measures to use, the correct value for each of the thresholds involved as well as the right operators to combine these measures are daunting tasks that can rarely be performed manually. Finding means to (help users) detect solutions that lead to correct links is thus of uttermost importance.

Consequently, the goal of this work is to present solutions to the *time-complexity* and the *accuracy challenges*. In addition, we give an overview of the framework in which these solutions were implemented and of use cases within which this framework was employed. We begin by giving an introduction to the link discovery problem and of the structure of this work in Part I. In Part II, we address the time-complexity challenge by presenting time-efficient algorithms for link discovery. We begin by presenting approaches that can be used in different affine spaces to compute links efficiently. We then address time efficiency for weighted measures before presenting approaches based on planning and parallel programming to further improve the scalability of link discovery. The aim of Part III is to address the second challenge. Here, we focus mainly on different types of machine learning approaches, including active learning, batch learning and unsupervised learning within settings such as genetic programming, gradient descent and refinement operators. Part IV presents the framework wherein the solutions described in Part II and Part III were implemented, i.e., the LIMES framework. Applications of the methods and approaches developed during the course of this work are presented in Part V, wherein we show the diverse areas within which the algorithms presented in this work have been employed. In the last part, Part VI, we reflect upon the approaches presented herein and present future work in this area of research.

1 See http://stats.lod2.eu/. Accessed July 19th, 2016.

PUBLICATIONS

The work presented in this thesis is the result of six years of intensive collaboration with other researchers and peers. It is thus almost entirely based on the following publications (in alphabetical order):

1. (Auer et al., 2013a): Auer, S., J. Lehmann, A.-C. Ngonga Ngomo, and A. Zaveri (2013). Introduction to linked data and its lifecycle on the web. In *Reasoning Web. Semantic Technologies for Intelligent Data Access - 9th International Summer School 2013, Mannheim, Germany, July 30 - August 2, 2013. Proceedings*, pp. 1–90. Springer.

2. (Capadisli et al., 2015): Capadisli, S., S. Auer, and A.-C. Ngonga Ngomo (2015). Linked SDMX data: Path to high fidelity statistical linked data. *Semantic Web 6*(2), 105–112.

3. (Gerber et al., 2013): Gerber, D., S. Hellmann, L. Bühmann, T. Soru, R. Usbeck, and A.-C. Ngonga Ngomo (2013). Real-time RDF extraction from unstructured data streams. In *The Semantic Web - ISWC 2013 - 12th International Semantic Web Conference, Sydney, NSW, Australia, October 21-25, 2013, Proceedings, Part I*, pp. 135–150.

4. (Hillner and Ngonga Ngomo, 2011): Hillner, S. and A.-C. Ngonga Ngomo (2011). Parallelizing LIMES for large-scale link discovery. In *Proceedings the 7th International Conference on Semantic Systems, I-SEMANTICS 2011, Graz, Austria, September 7-9, 2011*, pp. 9–16.

5. (Lehmann et al., 2012): Lehmann, J., T. Furche, G. Grasso, A.-C. Ngonga Ngomo, C. Schallhart, A. J. Sellers, C. Unger, L. Bühmann, D. Gerber, K. Höffner, D. Liu, and S. Auer (2012). deqa: Deep web extraction for question answering. In *The Semantic Web - ISWC 2012 - 11th International Semantic Web Conference, Boston, MA, USA, November 11-15, 2012, Proceedings, Part II*, pp. 131–147.

6. (Lyko et al., 2013): Lyko, K., K. Höffner, R. Speck, A.-C. Ngonga Ngomo, and J. Lehmann (2013). SAIM - one step closer to zero-configuration link discovery. In *The Semantic Web: ESWC 2013 Satellite Events - ESWC 2013 Satellite Events, Montpellier, France, May 26-30, 2013, Revised Selected Papers*, pp. 167–172.

7. (Morsey et al., 2011): Morsey, M., J. Lehmann, S. Auer, and A.-C. Ngonga Ngomo (2011). Dbpedia SPARQL benchmark - performance assessment with real queries on real data. In *The Semantic Web - ISWC 2011 - 10th International Semantic Web Conference, Bonn, Germany, October 23-27, 2011, Proceedings, Part I*, pp. 454–469. **Best Research Paper Award**

8. (Morsey et al., 2012): Morsey, M., J. Lehmann, S. Auer, and A.-C. Ngonga Ngomo (2012). Usage-centric benchmarking of RDF triple stores. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, July 22-26, 2012, Toronto, Ontario, Canada.*

9. (Nentwig et al., 2014): Nentwig, M., T. Soru, A.-C. Ngonga Ngomo, and E. Rahm (2014). Linklion: A link repository for the web of data. In *The Semantic Web: ESWC 2014 Satellite Events, Anissaras, Crete, Greece, May 25-29, 2014, Revised Selected Papers*, pp. 439–443.

10. (Ngonga Ngomo, 2011): Ngonga Ngomo, A.-C. (2011). A time-efficient hybrid approach to link discovery. In *Proceedings of the 6th International Workshop on Ontology Matching, Bonn, Germany, October 24, 2011*.

11. (Ngonga Ngomo, 2012a): Ngonga Ngomo, A.-C. (2012a). Link discovery with guaranteed reduction ratio in affine spaces with minkowski measures. In *The Semantic Web - ISWC 2012 - 11th International Semantic Web Conference, Boston, MA, USA, November 11-15, 2012, Proceedings, Part I*, pp. 378–393.

12. (Ngonga Ngomo, 2012b): Ngonga Ngomo, A.-C. (2012b). On link discovery using a hybrid approach. *J. Data Semantics 1*(4), 203–217.

13. (Ngonga Ngomo, 2013): Ngonga Ngomo, A.-C. (2013). ORCHID - reduction-ratio-optimal computation of geo-spatial distances for link discovery. In *The Semantic Web - ISWC 2013 - 12th International Semantic Web Conference, Sydney, NSW, Australia, October 21-25, 2013, Proceedings, Part I*, pp. 395–410.

14. (Ngonga Ngomo, 2014): Ngonga Ngomo, A.-C. (2014). HELIOS - execution optimization for link discovery. In *The Semantic Web - ISWC 2014 - 13th International Semantic Web Conference, Riva del Garda, Italy, October 19-23, 2014. Proceedings, Part I*, pp. 17–32.

15. (Ngonga Ngomo and Auer, 2011): Ngonga Ngomo, A.-C. and S. Auer (2011). LIMES - A time-efficient approach for large-scale link discovery on the web of data. In *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*, pp. 2312–2317.

16. (Ngonga Ngomo et al., 2014): Ngonga Ngomo, A.-C., S. Auer, J. Lehmann, and A. Zaveri (2014). Introduction to linked data and its lifecycle on the web. In *Reasoning Web. Reasoning on the Web in the Big Data Era - 10th International Summer School 2014, Athens, Greece, September 8-13, 2014. Proceedings*, pp. 1–99.

17. (Ngonga Ngomo et al., 2013): Ngonga Ngomo, A.-C., L. Kolb, N. Heino, M. Hartung, S. Auer, and E. Rahm (2013). When to reach for the cloud: Using parallel hardware for link discovery. In *The Semantic Web: Semantics and Big Data, 10th International Conference, ESWC 2013, Montpellier, France, May 26-30, 2013. Proceedings*, pp. 275–289. **Best Research Paper Award**

18. (Ngonga Ngomo et al., 2011): Ngonga Ngomo, A.-C., J. Lehmann, S. Auer, and K. Höffner (2011). RAVEN - active learning of link specifications. In *Proceedings of the 6th International Workshop on Ontology Matching, Bonn, Germany, October 24, 2011*.

19. (Ngonga Ngomo and Lyko, 2012): Ngonga Ngomo, A.-C. and K. Lyko (2012). EAGLE: efficient active learning of link specifications using genetic programming. In *The Semantic Web: Research and Applications - 9th Extended Semantic Web Conference, ESWC 2012, Heraklion, Crete, Greece, May 27-31, 2012. Proceedings*, pp. 149–163.

20. (Ngonga Ngomo and Lyko, 2013): Ngonga Ngomo, A.-C. and K. Lyko (2013). Unsupervised learning of link specifications: deterministic vs. non-deterministic. In *Proceedings of the 8th International Workshop on Ontology Matching co-located with the 12th International Semantic Web Conference (ISWC 2013), Sydney, Australia, October 21, 2013.*, pp. 25–36.

21. (Ngonga Ngomo et al., 2013): Ngonga Ngomo, A.-C., K. Lyko, and V. Christen (2013). COALA - correlation-aware active learning of link specifications. In *The Semantic Web: Semantics and Big Data, 10th International Conference, ESWC 2013, Montpellier, France, May 26-30, 2013. Proceedings*, pp. 442–456.

22. (Ngonga Ngomo et al., 2014): Ngonga Ngomo, A.-C., M. A. Sherif, and K. Lyko (2014). Unsupervised link discovery through knowledge base repair. In *The Semantic Web: Trends and Challenges - 11th International Conference, ESWC 2014, Anissaras, Crete, Greece, May 25-29, 2014. Proceedings*, pp. 380–394.

23. (Saleem et al., 2015): Saleem, M., Q. Mehmood, and A.-C. Ngonga Ngomo (2015). FEASIBLE: A Featured-Based SPARQL Benchmarks Generation Framework. In *The Semantic Web - ISWC 2015 - 14th International Semantic Web Conference, Bethlehem, Pennsylvania, USA*.

24. (Saleem et al., 2013): Saleem, M., S. S. Padmanabhuni, A.-C. Ngonga Ngomo, J. S. Almeida, S. Decker, and H. F. Deus (2013). Linked cancer genome atlas database. In *I-SEMANTICS 2013 - 9th International Conference on Semantic Systems, ISEM '13, Graz, Austria, September 4-6, 2013*, pp. 129–134. **Winner of I-Challenge 2013**

25. (Saleem et al., 2014) Saleem, M., M. R. Kamdar, A. Iqbal, S. Sampath, H. F. Deus, and A.-C. Ngonga Ngomo (2014). Big linked cancer data: Integrating linked TCGA and pubmed. *J. Web Sem. 27*, 34–41. **Winner of of Big Data Challenge 2014**

26. (Sherif and Ngonga Ngomo, 2015): Sherif, M. A. and A.-C. Ngonga Ngomo (2015). Semantic quran. *Semantic Web 6*(4), 339–345.

27. (Soru et al., 2015b): Soru, T., E. Marx, and A.-C. Ngonga Ngomo (2015). ROCKER: A refinement operator for key discovery. In *Proceedings of the 24th International Conference on World Wide Web, WWW 2015, Florence, Italy, May 18-22, 2015*, pp. 1025–1033.

28. (Soru and Ngonga Ngomo, 2013): Soru, T. and A.-C. Ngonga Ngomo (2013). Rapid execution of weighted edit distances. In *Proceedings of the 8th International Workshop on Ontology Matching co-located with the 12th International Semantic Web Conference (ISWC 2013), Sydney, Australia, October 21, 2013.*, pp. 1–12.

29. (Soru and Ngonga Ngomo, 2014): Soru, T. and A.-C. Ngonga Ngomo (2014). A comparison of supervised learning classifiers for link discovery. In *Proceedings of the 10th International Conference on Semantic Systems, SEMANTICS 2014, Leipzig, Germany, September 4-5, 2014*, pp. 41–44.

In each chapter of the thesis, we specify the publication(s) from which the content was selected and explicate the contributions of the author.

Every good gift and every perfect gift
is from above, and cometh down from the
Father of lights, with whom is no
variableness, neither shadow of turning.
James 1:17

ACKNOWLEDGMENTS

Writing acknowledgments for the longest piece of work I have ever written is a daunting task that is doomed to failure. No language of the world can really express my thankfulness to the large number of people who have made this work possible by direct as well as indirect means. Hence, while the following serves the role of acknowledgments, it is no more than a pale reflection of what my acknowledgments should and would be if I had an infinite amount of time to explicate them.

Many thanks go to Prof. Klaus-Peter Fähnrich, who gave me the freedom to pursue the diverse avenues of research that led to this and many other works. I am grateful for his wise advice pertaining to research, group management, and life in general. He provided me with invaluable insights behind the curtains of what appeared to be a mere analytical exercise. Similar thanks go to Prof. Sören Auer, who introduced me to what would become a passion, the Semantic Web. His works have heralded novel paths to data management and have served the whole research community well over the last years. In addition to being a superb research group lead, his altruistic and group-driven approach to addressing research challenges have had a great effect on me. My companion in lead, Prof. Jens Lehmann, also impacted my approach and methodology towards surfacing and dealing with the deeper questions that underlie what appeared to be mere toy problems. His great sense of humor has provided prolonged sessions of laughter when they were sorely needed and beyond.

I have been entrusted with the honor of leading a superb research group. Ingenious young brains confronted me with challenging research questions, multifarious answers akin to conundrums, difficult choices, insights into other ways of thinking, cultures and so much more. I do thank you all for all the time that we have spent together pondering, laughing, thinking, discussing and simply being.

My families, all perfect gifts, have provided me with unquantifiable support over all the years of my life. I do thank you from the bottom of my heart. You have taught me what sacrificial, unconditional and undeserved love really means (especially you my dear parents). Although I am far from any kind of perfection, you have loved and supported me over a lifetime, even when I was too busy to give back as you more than deserve. I thank God for surrounding me with not one but four of the most loving, caring, hilarious but also challenging families imaginable to a mere human. I love you all.

CONTENTS

LIST OF ALGORITHMS

Part I

PRELIMINARIES

The aim of the first part of this thesis is to provide the preliminaries necessary to understand it. We begin by giving an overview of the Linked Data Web and the principles upon which it is built (see Chapter 1). We then motivate this work out of the Linked Data principles and present the two major challenges which underlie the core problem addressed in this work, i.e., the *link discovery problem*. Given that link discovery can be modeled in many different fashions (e.g., as a supervised machine learning problem or as an optimization problem) when aiming at tackling specific parts of the problem, we refrain from giving a complete formal model of the problem at hand. Instead, we provide the formal specification of the aspect of link discovery tackled by the each of the different approaches presented in this work within the corresponding chapter. We complete this part with a chapter dedicated to explaining the structure of this thesis (see Chapter 2).

1 INTRODUCTION

1.1 MOTIVATION

Over the last three decades, the World Wide Web has developed into a global resource containing Exabytes of data generated around the globe.[1] It is now used daily by billions of individuals across the planet[2] and provides services and information which permeate virtually all areas of life.[3] One of the most common applications of the Web is the search for information[4] (Duggan, 2013; Smith and Page, 2015).[5] Very successful information portals, such as Wikipedia[6], provide bundled information about millions of entities. Still, while all the information necessary to answers questions, such as

```
Which Argentinian soccer players played in the Premier League?
```

can be found in Web resources, answering such questions is tedious when relying on current Web technologies (Auer et al., 2007). This is due to the pieces of information necessary to answer this query being distributed across several Web pages and thus being difficult to put together for a classical document-based search engine. The task at hand becomes even more difficult when trying to answer questions such as

```
Which are the side effects of drugs that cure diseases caused by
a mutation of the FOXP2 gene?
```

Here, the pieces of information necessary to answer the question are distributed across different Web sources such as DailyMed,[7] Drugbank[8] and Sider.[9]

The Semantic Web vision (Berners-Lee et al., 2001) aims to address the problem of information access on the Web by enabling machines to better process the content of the Web. In their seminal paper, Berners-Lee et al. (2001) describe a Semantic Web in which software agents can gather information from and across Web pages. Given the explicit semantics available to these agents, they are further able to integrate and fuse these pieces of information as well as answer queries based thereon. By these means, the agents envisioned can support humans in tasks such as booking and synchronizing schedules, answering complex questions, monitoring information sources and discovering relevant facts for applications. A number of languages have been developed over the last two decades to help implement this

1 http://www.worldwidewebsize.com/
2 http://www.internetlivestats.com/internet-users/
3 https://www.unternehmensregister.de
4 http://www.alexa.com/topsites
5 http://www.pewinternet.org/files/old-media//Files/Reports/2013/PIP_Cell%20Phone%20Activities%20May%202013.pdf
6 http://wikipedia.org
7 http://dailymed.nlm.nih.gov/dailymed/
8 http://www.drugbank.ca/
9 http://sideeffects.embl.de/

vision. For example, facts are represented as triples using RDF[10] (Resource Description Framework). RDFS[11] (Resource Description Framework Schema) is used for representing simple schemas without negation. OWL[12] (Web Ontology Language) goes a step further and allows representing complex ontologies for knowledge bases while SPARQL[13] (SPARQL Protocol and RDF Query Language)[14] allows querying the resulting knowledge bases.[15]

The current precursor of the Semantic Web is the Linked Data Web (also called Data Web or Web of Data) (Bizer et al., 2009; Auer et al., 2013b). While the Linked Data Web began as 12 interlinked knowledge bases in May 2007 (Ngonga Ngomo, 2011; Gerber and Ngonga Ngomo, 2012), it has now developed into a compendium of close to 10,000 datasets.[16] An excerpt of these datasets[17] is shown in Figure 1.1. The Linked Data Web is governed by the Linked Data principles,[18] which specify best practices for publishing data. These are:[19]

> **Principle 1**: "Use URIs as names for things."
>
> **Principle 2**: "Use HTTP URIs so that people can look up those names."
>
> **Principle 3**: "When someone looks up a URI, provide useful information, using the standards (RDF*, SPARQL)."
>
> **Principle 4**: "Include links to other URIs so that they can discover more things."

A large number of frameworks have been developed for transforming data to RDF (Principle 1, see Ngonga Ngomo et al. (2014) for an overview). Within these efforts, URI generation approaches such as Cool URIs[20] have been employed. Frameworks for exposing these URIs as HTTP URIs are also already commonly employed, including D2R (Eisenberg and Kanza, 2012),[21] SPARQLIFY (Stadler et al., 2015)[22] and Virtuoso.[23] These solutions also address the third principle.

In this work, we focus on the fourth principle, i.e., "Include links to other URIs, so that they can discover more things". Given a set S of RDF resources, the aim here is to generate new RDF statements that connect the resources from S with other RDF resources. This principle is of uttermost importance as it enables the paradigm change from data silos to interoperable data distributed across the Web. Hence, the inclusion of links plays a key role in applications such as question answering across several datasets (Bhagdev et al., 2008; Lopez et al., 2009; Shekarpour et al.,

10 http://www.w3.org/RDF/
11 http://www.w3.org/TR/2014/REC-rdf-schema-20140225/
12 http://www.w3.org/standards/techs/owl#w3c_all
13 http://www.w3.org/TR/sparql11-query/
14 SPARQL is a recursive acronym.
15 We refrain from delving into the details of these languages as they are beyond the scope of the work at hand. The keen reader is advised to follow the links provided. In the following, we will assume these languages to be known.
16 Data from http://stats.lod2.eu/, retrieved July 19th, 2016
17 Taken from http://lod-cloud.net/, snapshot created August 2014
18 http://www.w3.org/DesignIssues/LinkedData.html
19 All principles were copied from http://www.w3.org/DesignIssues/LinkedData.html, version of August 19th, 2015
20 http://www.w3.org/TR/cooluris/
21 http://d2rq.org/d2r-server
22 http://aksw.org/Projects/Sparqlify.html
23 http://virtuoso.openlinksw.com/

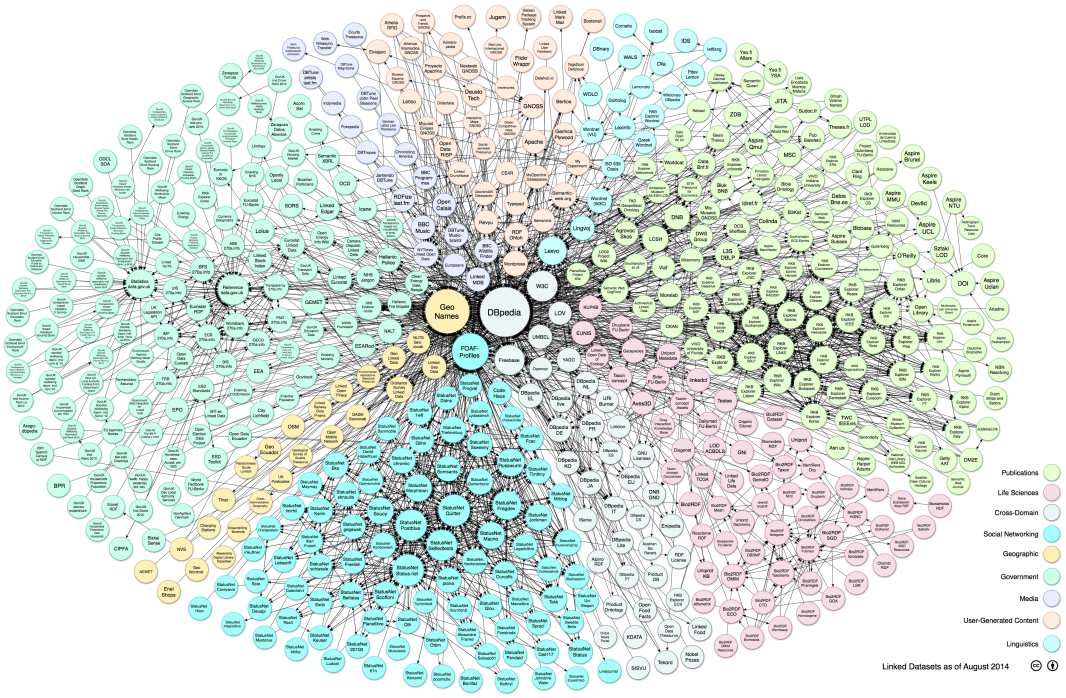


Figure 1.1: Excerpt of the Linked Open Data Cloud. State of August 2014.

2013), large-scale inferences (Urbani et al., 2010; McCusker and McGuinness, 2010), data integration for knowledge bases (Ma et al., 2009; Ben-David et al., 2010) and federated queries (Schmidt et al., 2011; Saleem and Ngonga Ngomo, 2014). For example, answering the query

```
List diseases whose possible drugs have no side effects
```

of the biomedical portion of the Question Answering on Linked Data (QALD) benchmark (version 4, see Unger et al. (2014)) requires making use of links between drugs described in DrugBank and drugs described in Sider. Moreover, making use of links between DBpedia and the New York Times is also necessary to answer the federated query CD1

```
Find all information about Barack Obama
```

from the federated query benchmark FedBench (Schmidt et al., 2011).

Several solutions could be envisaged to tackle the link discovery problem. First, one could try a manual approach. However, consider that, for example, DBpedia alone describes more than 3 million entities and is connected to LinkedGeoData, which describes more than 10 million resources. Creating links manually between these two knowledge bases would thus require humans to check the more than 3×10^{13} pairs of entities available across these two knowledge bases for possible links. Even if 1 billion humans were to partake in this endeavor and a check lasted only 1 minute, so would the billion persons need more than 10 weeks just for this knowledge base pair.[24] Now, with approximately 10^4 knowledge bases, we have circa 50 million pairs of knowledge bases on the Web of Data. Hence, a manual approach to link discovery is doomed to fail as it does not scale up to the size and number of knowledge bases on the Web of Data.

24 Assuming 8 hours of work per day and each pair being checked by one person.

1.2 GOAL

The considerations above lead to the main goal of this work, which is to devise *efficient means for the computation of accurate links between sets of resources*, i.e., efficient and accurate approaches for link discovery. One can envisage a plethora of different approaches towards achieving this goal, ranging from weighted logics (Droste and Gastin, 2009) to graph kernels (Gärtner et al., 2003). In this work, we focus on link discovery as tackled by *declarative approaches*. The basic formal model behind these approaches is as follows:

> Given a set S of source resources, a set T of target resources and a relation R, find the set $M = \{(s,t) \in S \times T : R(s,t)\}$.

The basic intuition behind declarative approaches is that finding M can be a tedious endeavor. Hence, declarative approaches aim to find a *similarity function* $\sigma : S \times T \to [0,1]$ and a *similarity threshold* θ such that the set $M' = \{(s,t,\sigma(s,t)) : \sigma(s,t) \geqslant \theta\}$ approximates M.[25] Under the declarative representation paradigm, two main challenges need to be addressed to allow for the *efficient* computation of *accurate links* between knowledge bases: the *time-complexity challenge* and the *accuracy challenge*.

The time-complexity challenge arises from the specification of the set M'. Naïve approaches towards computing the set of all $(s,t,\sigma(s,t))$ in M' would require $|S| \times |T|$ computations of σ, i.e., they would be quadratic in complexity. While such computations can be carried out in acceptable times for small S and small T, the execution time of naïve implementations reaches weeks when aiming to compare large knowledge bases. For example, if a similarity computation were to last 1 ms, S were DBpedia (3×10^6 resources) and T were LinkedGeoData (10^7 million resources), then the computation of M' on a single-core CPU would last more than 950 years. Such runtimes are clearly impractical as they go beyond the life expectancy at birth of a 21st-century human. While using modern hardware architectures can alleviate the problem, even an ideal (speedup = number of processors) parallel implementation of a naïve approach run on a cluster with 500 processors would still need close to two years to terminate. There is thus the clear need for more approaches that allow for computing all the elements of the set M' in less time than naïve approaches. Part II of this work deals with this challenge.

The accuracy challenge is also a direct consequence of the definition of M'. While this definition makes clear that a similarity measure σ and a similarity threshold θ are needed, the means to find these two parameters are not specified. In general, σ is a composite (also called complex) function and is thus difficult to devise manually. For example, finding resources that stand for the same real-world entity across movie directors in DBpedia and LinkedMDB (Linked Movie Database[26]) demands the combination of string similarity measures such as trigrams, cosine and jaccard[27] as defined in Xiao et al. (2008) (see Figure 1.2). We devise machine-learning approaches to address this challenge, which we present in Part III. These approaches abide by paradigms as various as unsupervised (Nikolov et al., 2012; Ngonga Ngomo et al., 2014), supervised (Ngonga Ngomo

25 Equivalent formulations of the problem with distance measures can found in the literature and have been considered by the author, e.g., in (Ngonga Ngomo et al., 2013; Ngonga Ngomo, 2011, 2012b, 2013). Extended formulations of the problem statement can also be found in the subsequent chapters.

26 http://www.linkedmdb.org/

27 Results obtained with the EAGLE algorithm, see Chapter 12.

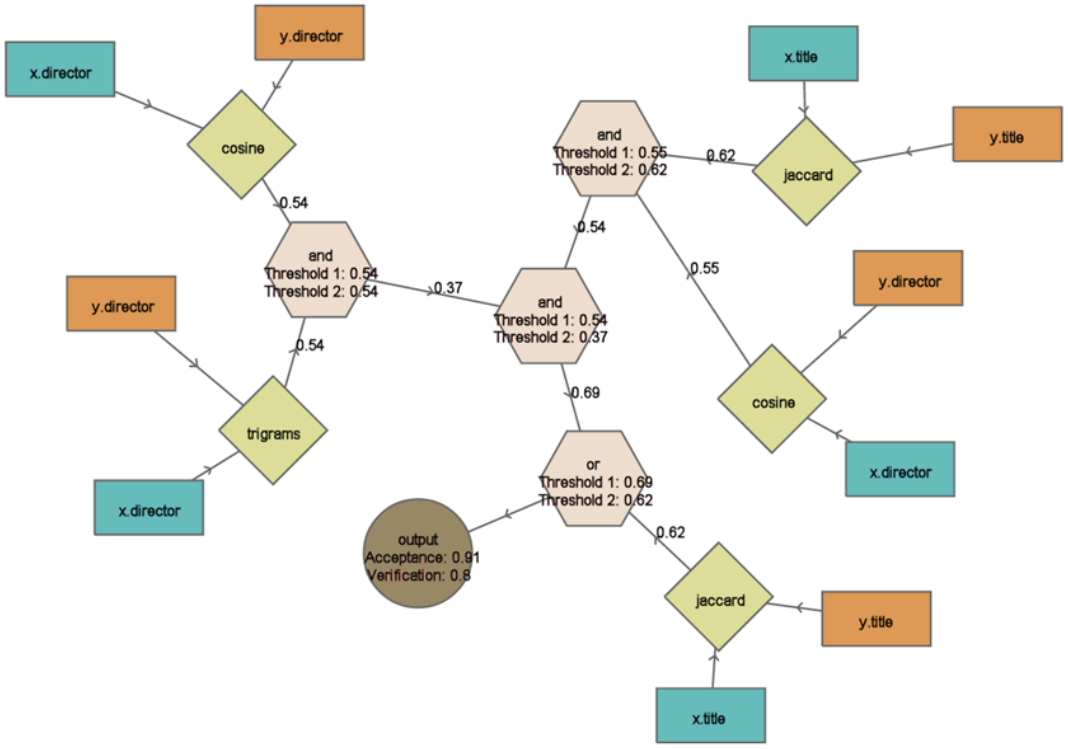


Figure 1.2: Graphical representation of a composite similarity measure. The figure is aimed to show the complexity of composite measures needed to discover links with high accuracies. Formally, the figure displays a tree that is to be read from the leaves to the root. The exact semantics of this tree are defined in the subsequent chapters.

and Lyko, 2013) and active machine learning (Settles, 2012; Ngonga Ngomo et al., 2011; Ngonga Ngomo and Lyko, 2012).

The secondary goal of this work is to exemplify how the algorithms underlying link discovery approaches can be used in various applications. In particular, we use some of our insights to ensure the scalability of benchmark generators based on real queries and real data (Morsey et al., 2011, 2012; Saleem et al., 2015). This family of benchmarks completes previous works on synthetic benchmarking for triple stores (e.g.,(Bizer and Schultz, 2009)). Open Knowledge Extraction (Gerber et al., 2013) as well as domain-specific question answering (Lehmann et al., 2012) are also shown to profit from our insights. The solutions presented in this work are integrated into the LIMES framework.[28]

28 http://limes.sf.net

2 STRUCTURE

2.1 INTRODUCTION

The structure of this thesis reflects the two core research topics underlying this work. Over the last six years, the author has collaborated intensively with several researchers while aiming to address these topics. The resulting collection of publications build the essence of this work. Most of the chapters presented herein are based on at least one peer-reviewed scientific publication. The publications as well as the contributions of the author are clearly mentioned in the preamble of each chapter. The author took great care to mark the source of the content from which this work originates as clearly as possible. Should there however be any unintentionally unmarked source, so would the author be pleased to produce an erratum stating this error explicitly. The author chose to write this work in the "we" form.

Given the large body of publications, the difference in formal modelling and the time frame within which the publications were written, this work adopts a decentral structure. Each chapter has:

- its *own introduction*, which describes the motivation behind the chapter according to the state of the art at the time of writing,
- its *own related work section*, which summarizes the work related to the approach presented in the chapter at the time of publication of the corresponding paper(s),
- its *own evaluation section*, in which the approach(es) presented in the chapter are evaluated against other approaches that represented the state of the art at the time of writing and
- its *own conclusions*, which aim to give the reader some insights pertaining to the state of research and the further targeted developments of the approaches presented in the chapter at the time of publication.

It is most certainly already evident to the reader that the work is subdivided into parts. In the following, we present the content of each of these parts in more detail and give an overview of the chapters they contain.

2.2 PART I: PRELIMINARIES

In Part I, we began by presenting the context of this work as well as the motivation behind it (see Chapter 1). This motivation led us to pose two core questions pertaining to the *scalability* and the *accuracy of link discovery*. Providing approaches to address aspects of these questions is the quintessence of this work. In Chapter 2, we present the structure of this work.

2.3 PART II: RUNTIME EFFICIENCY

The second part of this work is devoted to approaches for improving the runtime of link discovery. In Chapter 3, we present the LIMES algorithm—one of the first approaches dedicated to the scalability of link discovery. The approach uses the triangle inequality to efficiently portion affine spaces and improve the runtime of link discovery within these spaces. Chapter 4 builds upon this idea and focuses on improving link discovery by using an equidistant portioning of an affine space to improve the runtime of bounded distances such as the Euclidean distance. The idea of portioning affine spaces is extended in Chapter 5, where spaces with Minkowski distances are considered. Here, the author develops the first reduction-ratio-optimal approach for link discovery. A different type of space is considered in Chapter 6, where the author presents a reduction-ratio-optimal approach for orthodromic spaces. The rapid execution of bounded distances can obviously also be carried out when dealing with strings. While Chapter 3 deals with the edit distance, Chapter 7 deals with weighted edit distances and relies on a different, filter-based paradigm to improve their scalability.

In addition to benefiting from the improvement of the runtime of single similarity measures, the runtime of link discovery can also be improved by a better planning of the execution of link specifications as well as by running link specifications in parallel. Chapter 8 presents the first execution planner for link discovery and shows how orders of magnitude of runtime improvement can be achieved by an intelligent combination of runtime and scalability prediction approaches. The final chapters of this work address another concept for improving the runtime of link discovery, i.e., the execution of link discovery tasks on parallel hardware. Chapter 9 shows how the LIMES algorithm presented in Chapter 3 can be implemented in parallel using the map-reduce paradigm. The subsequent and last chapter of this section, Chapter 10, delves deeper into different means of parallel implementations for link discovery (cloud-based, graphics processing units and thread-based) and provides a comparison of these approaches. The work presented in this final chapter was deemed worthy of the *best research paper* award at the Extended Semantic Web Conference 2013.

2.4 PART III: LEARNING LINK SPECIFICATIONS

While Part II focuses on the scalability of link discovery, the third part of this work is centred on the accuracy of link discovery. In particular, machine-learning approaches for link specifications are at the core of the methods presented herein. This part begins with Chapter 11, which presents the first active learning approach for link discovery dubbed RAVEN. The algorithm relies on a paradigm akin to the perceptron algorithm to detect highly informative link candidates and gather corresponding annotations from the end user. A different active learning paradigm is the main subject in Chapter 12, where the genetic-programming-based algorithm EAGLE is presented and evaluated. An extension of EAGLE towards a better selection of most informative link candidates is presented in Chapter 13, where the COALA algorithm is presented. Different twists towards learning link specification are taken in the subsequent chapters. In Chapter 14, unsupervised machine learning approaches for link discovery are analysed. We also consider learning specifications on more than two knowledge bases concurrently in an unsupervised manner

in Chapter 15. Chapter 16 presents a brief comparison of classical machine-learning approaches on the link discovery task while the concluding chapter, Chapter 17, presents a method for improving the scalability of link discovery by means of keys for linking.

2.5 PART IV: THE LIMES FRAMEWORK

The approaches presented above were all implemented within the LIMES framework (not to be confused with the LIMES algorithm presented in Chapter 3). The idea behind this framework was to develop a highly time-efficient link discovery framework that would allow end users to link large knowledge bases easily. We begin by giving the reader some insights pertaining to how to use the LIMES framework (in Chapter 18) as well as one of its graphical user interfaces (in Chapter 19). Finally, we briefly describe a repository for storing and managing the link resulting from LIMES or any other link discovery framework in Chapter 20.

2.6 PART V: APPLICATIONS

The author's work on link discovery has had several scientific and commercial applications. In this part of the work, we present an excerpt of these applications. In Chapter 21, we begin by presenting an application that won the *best research paper award* at the International Semantic Web Conference 2011. Here, the LIMES framework was used to rapidly compute the similarity of queries to generated benchmarks out of query logs. An even more recent work in this direction is presented in Chapter 22, where the exemplar-based space portioning approach underlying the original LIMES algorithm is reused for generating even more accurate benchmarks for federated queries. Another area in which the LIMES framework was used successfully is question answering. In Chapter 23, we show how this framework was used to rapidly compute links between geo-spatial resources in and around the city of Oxford, England so as to empower the question answering framework DEQA. The LIMES framework was also used in the open knowledge extraction framework RdfLiveNews, which is described in Chapter 24. There, our framework was used to detect similar natural-language expressions, which were later used as labels for automatically discovered predicate. This section is concluded by three chapters which present datasets which used LIMES for the purpose for which it was primarily designed, i.e., linking to other knowledge bases. Chapter 25 relies on our framework for linking bio-medical data to reference knowledge bases. Chapter 26 does the same for statistical data represented in the Statistical Data and Metadata Exchange format. Finally, Chapter 27 shows how LIMES can be used to link linguistics knowledge bases.

2.7 PART VI: CONCLUSIONS

In this section, we begin by summarizing the results achieved over the last years. An inspection of these results and a discussion of possible further directions for the work presented herein follow. We conclude this work with these insights.

Part II

RUNTIME EFFICIENCY

The second part of this thesis focuses on the first challenge behind Link Discovery, i.e., the inherent runtime complexity of Link Discovery. We present a series of approaches that address this problem by three different means. First, we present approaches that aim at improving of the runtime of bounded measures (Chapter 3–Chapter 7). Thereafter, we focus on planning and present the first planner for link discovery (Chapter 8). Finally, we delve into approaches for the parallel execution of link discovery approaches (Chapter 9–Chapter 10)

3 THE LIMES APPROACH

PREAMBLE

This chapter is based on (Ngonga Ngomo and Auer, 2011) and was one of the first papers that focused on scalability in link discovery. The author designed, implemented and evaluated the algorithm presented herein.

3.1 INTRODUCTION

The core idea behind the Linked Data paradigm is to facilitate the transition from the document-oriented Web to the Semantic Web by extending the Web with a data commons consisting of interlinked data sources (Bizer et al., 2009; Volz et al., 2009). While the number of triples in data sources across the Linked Open Data Cloud increases steadily and has surpassed 25 billion,[1] links still constitute less than 5% of the total number of triples available on the Linked Data Web. In addition, while the number of tools for publishing Linked Data on the Web grows steadily, there is a significant lack of time-efficient solutions for discovering links between these datasets. Yet, links between knowledge bases play a key role in important tasks such as cross-ontology question answering (Lopez et al., 2009), large-scale inferences (Urbani et al., 2010) and data integration (Ben-David et al., 2010).

To carry out a matching task, the distance measure as defined by the user is usually applied to the value of some properties of instances from S and T so as to detect instances that should be linked. Instances whose distance is lower or equal to a given threshold are considered to be candidates for linkage. The a-priori complexity of a matching task is proportional to $|S||T|$, an unpractical proposition as soon as the source and target knowledge bases become large. For example, discovering duplicate cities in DBpedia (Auer et al., 2007) alone would necessitate approximately 0.15×10^9 distance computations. Hence, the provision of time-efficient approaches for the reduction of the time complexity of link discovery is a key challenge for the Linked Data community.

In this chapter, we present LIMES (Link Discovery Framework for metric spaces)—a time-efficient approach for the discovery of links between Link Data sources. LIMES addresses the scalability problem of link discovery by using the *triangle inequality* in metric spaces to compute pessimistic estimates of instance similarities. Based on these approximations, LIMES can filter out a large number of instance pairs that cannot suffice the matching condition set by the user. The real similarities of the remaining instance pairs are then computed and the matching instances are returned. We show that LIMES requires a significantly smaller number of *comparisons* than brute-force approaches by using synthetic data. In addition, we show that our approach is superior to state-of-the-art link discovery frameworks by comparing their runtime in real-world use cases.

The contributions of this chapter are as follows:

1 http://www4.wiwiss.fu-berlin.de/lodcloud/ accessed am 03.01.2011.

- We present the lossless and time-efficient approach for the large-scale matching of instances in metric spaces dubbed LIMES.
- We present two novel algorithms for the efficient approximation of distances within metric spaces based on the triangle inequality.
- We evaluate LIMES on synthetic data by using the number of comparisons necessary to complete the given matching task. Furthermore, we evaluate our approach on real data against the SILK framework (Volz et al., 2009) with respect to the runtime of both frameworks on the same hardware.

The remainder of this chapter is structured as follows: after reviewing related work in Section 3.2 we develop the mathematical framework underlying LIMES in Section 3.3. We present the LIMES approach in Section 3.4 and report on the results of an experimental evaluation in Section 3.5. We conclude with a discussion and an outlook on future work in Section 3.6.

3.2 RELATED WORK

Using the triangle inequality for improving the runtime of algorithms is not a novel idea. This inequality has been used for tasks such as data clustering (Cilibrasi and Vitányi, 2005), spatial matching (Wong et al., 2007) and query processing (Yao et al., 2003). Yet, to the best of our knowledge it has never been used previously for the discovery of links on the Web of Data.

Current frameworks for link discovery on the Web of Data can be subdivided into two categories: *domain-specific* and *universal* frameworks. Domain-specific link discovery frameworks aim at discovering links between knowledge bases from a particular domain. For example, the RKB knowledge base (RKB-CRS) (Glaser et al., 2009) computes links between universities and conferences while Another domain-specific tool is GNAT (Raimond et al., 2008), which can be used to discover links between music datasets. GNAT uses similarity propagation on audio fingerprinting to discover links between music datasets.

Universal link discovery frameworks are designed to carry out mapping tasks independently from the domain of the source and target knowledge bases. For example, RDF-AI (Scharffe et al., 2009) implements a five-step approach that comprises the preprocessing, matching, fusion, interlink and post-processing of datasets. These modules can be configured by means of XML-files. Like LIMES, SILK (Volz et al., 2009) is a time-optimized for link discovery. Instead of using the characteristics of metric spaces, SILK uses rough index pre-matching to reach a quasi-linear time-complexity. The drawback of the pre-matching approach is that the recall of this framework is not guaranteed to be 1. SILK allows the manual configuration of data blocks to minimize the runtime of the matching process. It can be configured by using the SILK-Link Specification Language, which is based on XML.

The task of discovering links between knowledge bases is closely related with record linkage (Elmagarmid et al., 2007; Winkler, 2006) and deduplication (Bleiholder and Naumann, 2008). The database community has produced a vast amount of literature on efficient algorithms for solving these problems. Different blocking techniques such as standard blocking, sorted-neighborhood, bigram indexing, canopy clustering and adaptive blocking (see, e.g., (Köpcke et al., 2009)) have been developed to address the problem of the quadratic time complexity of brute-force

comparison methods. The idea is to filter out obvious non-matches efficiently before executing the more detailed and time-consuming comparisons.

The difference between the approaches described above and our approach is that LIMES uses the triangle inequality to portion the metric space. Each of these portions of the space is then represented by an exemplar (Frey and Dueck, 2007) that allows to compute an accurate approximation of the distance between each instance in this region and others instances. By these means, we can discovery links between Linked Data sources efficiently.

3.3 MATHEMATICAL FRAMEWORK

In this section, we present the mathematical principles underlying the LIMES framework. We present the formal definition of a matching task within metric spaces. Then, we use this definition to infer upper and lower boundary conditions for distances based on the triangle inequality. Finally, we show how these boundary conditions can be used to reduce the number of comparisons necessary to complete a mapping.

3.3.1 *Preliminaries*

In the remainder of this chapter, we use the following notation. Let A be an affine space. m, m_1, m_2, m_3 symbolize metrics on A; x, y and z represent points from A and α, β, γ and δ are scalars, i.e., elements of $\mathbb{R}$. Furthermore, we assume that (A, m) is a metric space.

Definition 1 (Matching task)**.** *Given two sets S (source) and T (target) of instances, a metric m and a threshold $\theta \in [0, \infty[$, the goal of a matching task is to compute the set M of triples (i.e., the mapping) $(s, t, m(s, t))$ of all instances $s \in S$ and $t \in T$ such that $m(s, t) \leqslant \theta$.*

We call each computation of the distance $m(s, t)$ a *comparison*. The time complexity of a mapping task can be measured by the number of comparisons necessary to complete this task. A-priori, the completion of a matching task requires $O(|S||T|)$ comparisons. In this chapter, we show how the number of comparisons necessary to map two knowledge bases can be reduced significantly by using the mathematical characteristics of metric spaces. For this purpose, we make particularly use of the triangle inequality (TI) that holds in metric spaces.

3.3.2 *Distance Approximation Based on the Triangle Inequality*

Given a metric space (A, m) and three points x, y and z in A, the TI entails that

$$m(x, y) \leqslant m(x, z) + m(z, y). \tag{3.1}$$

Without restriction of generality, the TI also entails that

$$m(x, z) \leqslant m(x, y) + m(y, z), \tag{3.2}$$

thus leading to the following boundary conditions in metric spaces:

$$m(x, y) - m(y, z) \leqslant m(x, z) \leqslant m(x, y) + m(y, z). \tag{3.3}$$

Equation 3.3 has two major implications. The first is that the distance from a point x to any point z in a metric space can be approximated when knowing the distance from x to a reference point y and the distance from the reference point y to z. We call such a reference point an *exemplar* (following (Frey and Dueck, 2007)). The role of an exemplar is to be used as a sample of a portion of the metric space A. Given an input point x, knowing the distance from x to an exemplar y allows to compute lower and upper bounds of the distance from x to any other point z at a known distance from y.

The second implication of Equation 3.3 is that the real distance from x to z can only be smaller than θ if the lower bound of the approximation of the distance from x to z via *any exemplar* y is also smaller than θ. Thus, if the lower bound of the approximation of the distance $m(x, z)$ is larger than θ, then $m(x, z)$ itself must be larger than θ. Formally,

$$m(x, y) - m(y, z) > \theta \Rightarrow m(x, z) > \theta. \tag{3.4}$$

Supposing that all distances from instances $t \in T$ to exemplars are known, reducing the number of comparisons simply consists of using Equation 3.4 to compute an approximation of the distance from all $s \in S$ to all $t \in T$ and computing the real distance only for the (s, t) pairs for which the first term of Equation 3.4 does not hold. This is the core of the approach implemented by LIMES.

3.4 APPROACH

In this section, we present the LIMES approach in more detail. First, we give an overview of the workflow it implements. Thereafter, we present the core algorithms underlying our approach.

3.4.1 *Overview*

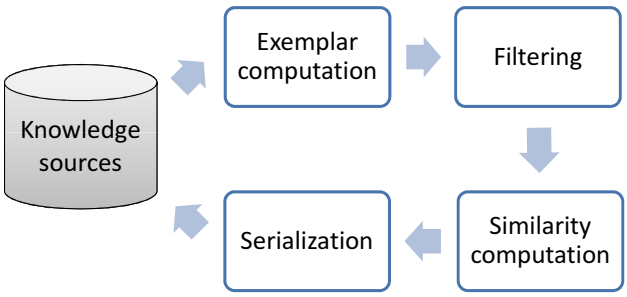


Figure 3.1: General Workflow of LIMES

The general workflow implemented by LIMES comprises four steps (as depicted in Figure 3.1). Given the source S, the target T and the threshold θ, LIMES first computes a set E exemplars for T (step 1). This process is concluded by matching each point $t \in T$ to the exemplar closest to it. In step 2 and 3, the matching *per se* is carried out. For each $s \in S$ and each $e \in E$, the distance $m(s, e)$ is computed. This distance is used to approximate the distance from s to every $t \in T$ (step 2). We call this step *filtering*. The filtering step implements the central innovation of our approach. The main advantage here is that since pessimistic estimates are used,

it is guaranteed to lead to exactly the same mapping as a brute-force approach while at the same time reducing the number of comparisons significantly. After the filtering, the real distance between the remaining $s \in S$ and the $t \in T$ for which the first term of Equation 3.4 did not hold are computed (step 3). Finally, the mappings $(s, t, m(s, t))$ such that $m(s, t) \leqslant \theta$ are serialized, i.e., written in a user-defined output stream according to a user-specified format, e.g. in an NTriples file.[2] (step 4)

3.4.2 *Computation of Exemplars*

The role of exemplars is to represent a portion of a metric space. Accordingly, the best distribution of exemplars should achieve a homogeneous portioning of this metric space. The core idea underlying the computation of exemplars in LIMES is to select a set of exemplars in the metric space underlying the matching task in such a way that they are distributed in a uniform way in the metric space. One way of achieving this goal is by ensuring that the exemplars are very dissimilar, i.e., very distant from each other. The approach we use to generate such exemplars is shown in Algorithm 3.1.

Algorithm 3.1 Computation of exemplars

Require: Number of exemplars n, target knowledge base T
Ensure: Set E of exemplars and their mapping to the instances in T
1: Pick random point $e_1 \in T$
2: Set $E = E \cup \{e_1\}$, $\eta = e_1$
3: Compute the distance from e_1 to all $t \in T$
4: **while** $|E| < n$ **do**
5: Get a random point e' such that $e' \in \operatorname{argmax}_t \sum_{t \in T} \sum_{e \in E} m(t, e)$
6: $E = E \cup \{e'\}$;
7: Compute the distance from e' to all $t \in T$
8: **end while**
9: Map each point in $t \in T$ to one of the exemplars $e \in E$ such that $m(t, e)$ is minimal
10: **return** E

Let n be the desired number of exemplars and E the set of all exemplars. We initialize E by picking a random point e_1 in the metric space (T, m) and setting $E = \{e_1\}$ (see Algorithm 3.1 lines 1-2). Then, we compute the distance from the exemplar e_1 to every other point in T (line 3). As long as the size of E has not reached n, we iterate lines 5 to 7: In line 5, we pick a point $e' \in T$ such that the sum of the distances from e' to the exemplars $e \in E$ is maximal (there can be many of these points). This point is chosen as new exemplar and added to E (line 6). Then, we compute the distance from e' to all other points in T (line 7).

Once E has reached the size n, we terminate the iteration. Finally, we map each point to the exemplar to which it is most similar (line 9) and terminate (line 10). This algorithm has a constant time complexity of $O(|E||T|)$.

2 http://www.w3.org/2001/sw/RDFCore/ntriples/

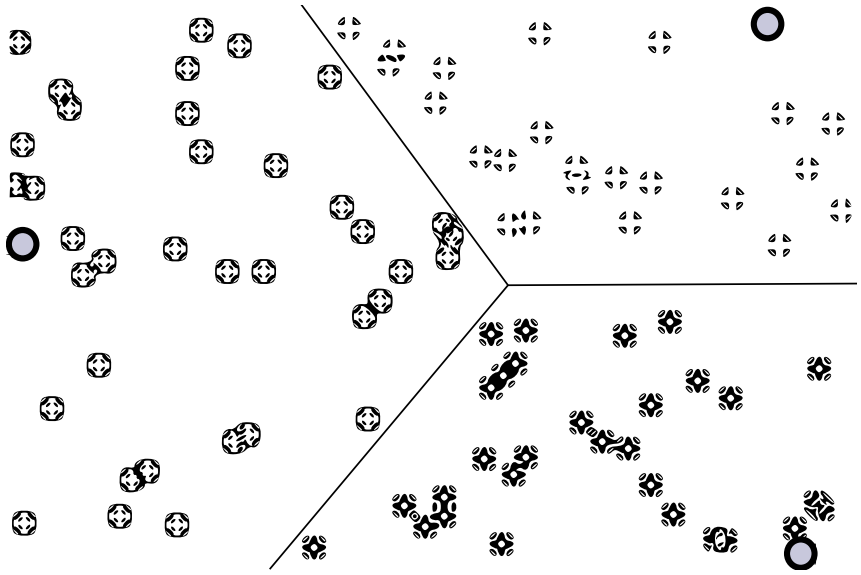


Figure 3.2: Mapping of points to three exemplars in a metric space. The exemplars are displayed as gray disks.

3.4.3 *Matching Based on Exemplars*

The instances associated with an exemplar $e \in E$ in line 9 of Algorithm 3.1 are stored in a list L_e sorted in descending order with respect to their distance to e. Let $\lambda_1^e ... \lambda_m^e$ be the elements of the list L_e. The goal of matching an instance s from a source knowledge base to a target knowledge base w.r.t. a metric m is to find all instances t of the target knowledge source such that $m(s,t) \leqslant \theta$, where θ is a given threshold. The matching algorithm based on exemplars is shown in Algorithm 3.2.

We only carry out a comparison when the approximation of the distance is less than the threshold. We terminate the similarity computation for an exemplar e as soon as the first λ^e is found such that the lower bound of the distance is larger than θ. This break can be carried out because the list L_e is sorted, i.e., if $m(s,e) - m(e,\lambda_i^e) > \theta$, then the same inequality holds for all λ_j^e with $j > i$. In the worst case, our matching algorithm has the time complexity $O(|S||T|)$, leading to a total worst-time complexity of $O((|E| + |S|)|T|)$, which is larger than that of brute-force approaches. However, as our evaluation with both synthetic and real data shows, a correct parameterization of LIMES leads to significantly smaller numbers of comparisons and runtimes.

3.5 EVALUATION

In this section, we elucidate the following four central evaluation questions:

Q_1: *What is the best number of exemplars?*

Q_2: *What is the relation between the threshold θ and the total number of comparisons?*

Q_3: *Does the assignment of S and T matter?*

Q_4: *How does* LIMES *compare to other approaches?*

To answer Q_1 to Q_3, we performed an evaluation on synthetic data as described in the subsequent section. Q_4 was elucidated by comparing the runtimes of LIMES and SILK on three different real-world matching tasks.

Algorithm 3.2 Computation of mappings

Require: Set of exemplars E, point s, metric m, threshold θ
Ensure: Mapping M for s

```
M = ∅
for e ∈ |E| do
    if m(s, e) ⩽ θ then
        M = M ∪ {e}
    end if
    for i = 1...|L_e| do
        if (m(s, e) − m(e, λ_i^e)) ⩽ θ then
            if m(s, λ_i^e) ⩽ θ then
                M = M ∪ {λ_i^e}
            else
                break;
            end if
        end if
    end for
end for
return M
```

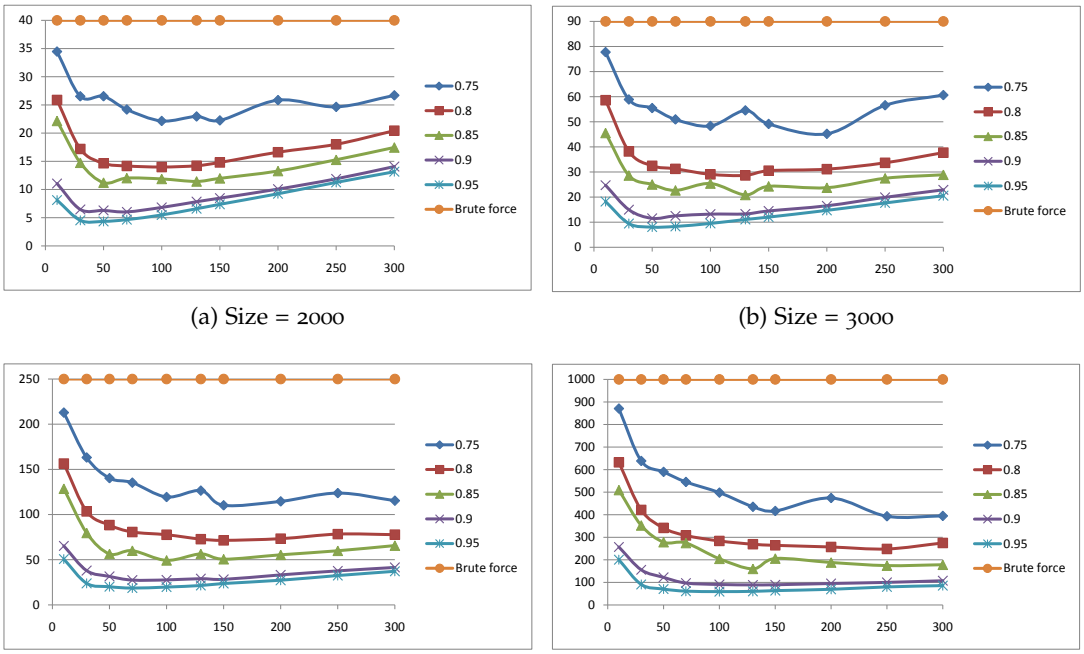
3.5.1 *Evaluation with Synthetic Data*

The general experimental setup for the evaluation on synthetic data was as follows: The source and target knowledge bases were filled with random strings having a maximal length of 10 characters. We used the Levenshtein metricto measure string similarity. Each of the matching tasks was carried out five times and we report average values in the following.

To address the first Q_1 and Q_2, we considered four matching tasks on knowledge bases of sizes between $2,000$ and $10,000$. We varied the thresholds between 0.95 and 0.75 and the number of exemplars between 10 and 300. We measured the average number of comparisons necessary to carry out each of the matching tasks (see Figure 3.3). Two main conclusions can be inferred from the results. First, the results clearly indicate that the best number of exemplars diminishes when the similarity threshold θ is increased. In general, the best value for $|E|$ lies around $\sqrt{|T|}$ for $\theta \geqslant 0.9$, which answers Q_1. The relation between θ and $|E|$ is a direct cause of our approach being based on the triangle inequality. Given a high threshold, even a rough approximation of the distances is sufficient to rule out a significant number of target instances as being similar to a source instance. However, a low threshold demands a high number of exemplars to be able to rule out a significant number of target instances.

An analysis of the results displayed in Figure 3.3 also allows to answer Q_2. The higher the value of θ, the smaller the number of comparisons. This is due to the stricter filtering that results from the higher threshold and consequently leads to a smaller number of required comparisons. An important observation is that, the larger the size of the knowledge bases S and T, the higher the speedup obtained by using the LIMES approach. For example, while LIMES necessitates approximately 7 times less comparisons than a brute-force approach for the knowledge bases of

size 2,000 and $\theta = 0.95$ in the best case, it requires approximately 17 times less comparisons for knowledge bases of size 10,000 with the same threshold settings.



(a) Size = 2000 (b) Size = 3000

(c) Size = 5000 (d) Size = 10000

Figure 3.3: Comparisons required by LIMES for different numbers of exemplars on knowledge bases of different sizes. The x-axis shows the number of exemplars, the y-axis the number of comparisons in multiples of 10^5.

To address Q_3, we measured the average number of comparisons required to map synthetic knowledge bases of sizes between 1,000 and 10,000 in all possible combinations of sizes for S and T. For this experiment, the number of exemplars was set to $\sqrt{|T|}$. θ was set to 0.9. The results of this experiment are summarized in Table 3.1.

Overall, the experiment shows that whether source or target knowledge base is larger does not affect the number of comparisons significantly. It appears that the results are slightly better when $|T| \leqslant |S|$. Yet, the difference between the number of comparisons lies below 5% in most cases and is thus not significant. Therefore, the link discovery can always be carried out by simply following the specification of the user with respect to which endpoint is the source resp. the target for the matching task.

3.5.2 *Evaluation with Real Data*

To answer the question Q_4, we evaluated the performance of LIMES on real data by comparing its runtime with that of the (to the best of our knowledge) only time-optimized link discovery framework SILK. Non-optimized frameworks would perform like a brute-force approach, which is clearly inferior to LIMES. To ensure an objective comparison of the runtimes, we only considered the time necessary for both frameworks to carry out the comparisons in our evaluation . Each of the time measurements was carried out three times and only the best runtime was considered. Every time measurement experiment was carried out as a single thread on a

	1000	2000	3000	4000	5000	6000	7000	8000	9000	10000
1000	0.20	0.37	0.53	0.69	0.88	1.04	1.14	1.40	1.58	1.67
2000	0.36	0.64	0.88	1.24	1.37	1.63	1.97	2.25	2.50	2.70
3000	0.51	0.86	1.17	1.57	2.00	2.09	2.69	2.91	3.35	3.58
4000	0.70	1.11	1.59	2.00	2.45	2.88	3.10	3.61	3.94	4.50
5000	0.85	1.36	1.87	2.28	2.81	3.39	3.91	4.20	4.84	5.54
6000	1.02	1.60	2.14	2.81	3.29	3.93	4.44	4.96	5.39	6.08
7000	1.22	1.86	2.58	3.15	3.66	4.35	5.11	5.69	6.44	6.62
8000	1.41	2.04	2.78	3.43	4.06	4.98	5.51	6.55	7.14	7.53
9000	1.63	2.36	2.99	3.85	4.72	5.44	6.25	6.88	7.59	8.20
10000	1.80	2.62	3.51	4.25	4.97	6.01	6.33	7.81	8.31	9.15

Table 3.1: Average number of comparisons (in millions) for matching knowledge bases of different sizes. The columns are the size of the source knowledge base, while the row are the size of the target knowledge base.

32-bit system with a 2.5GHz Intel Core Duo CPU and 4GB RAM. For our experiments, we used version 0.3.2 of LIMES and version 2.0 of SILK. The number of exemplars for LIMES was set to $\sqrt{|T|}$.

	Drugs	**SimCities**	**Diseases**
$\|S\|$	4,346	12,701	23,618
$\|T\|$	4,772	12,701	5,000
$\|E\|$	69	112	70
Source	DBpedia	DBpedia	MESH
Target	Drugbank	DBpedia	LinkedCT

Table 3.2: Overview of runtime experiments. $|S|$ is the size of the source knowledge base, $|T|$ is the size of the target knowledge base and $|E|$ is the number of exemplars used by LIMES during the experiment.

The experiments on real data were carried out in three different settings as shown in Table 3.2. The goal of the first experiment, named Drugs, was to map drugs in DBpedia[3] and Drugbank[4] by comparing their labels. The goal of the second experiment, named SimCities, was to detect duplicate cities within DBpedia by comparing their labels. The purpose of the last experiment, named Diseases, was to map diseases from MESH[5] with the corresponding diseases in LinkedCT[6] by comparing their labels.

Figure 3.4 shows a relative comparison of the runtimes of SILK and LIMES. The absolute runtimes are given in Table 3.3. LIMES outperforms SILK in all experimental settings. It is important to notice that the difference in performance grows with the (product of the) size of the source and target knowledge bases. While

3 http://dbpedia.org/sparql
4 http://www4.wiwiss.fu-berlin.de/drugbank/sparql
5 http://mesh.bio2rdf.org/sparql
6 http://data.linkedct.org/sparql

	LIMES					SILK
	0.95	0.9	0.85	0.8	0.75	
Drugs	86	120	175	211	252	1,732
SimCities	523	979	1,403	1,547	1,722	33,786
Diseases	546	949	1,327	1,784	1,882	17,451

Table 3.3: Absolute runtimes of LIMES and SILK. All times are given in seconds. The values in the second row of the table are the similarity thresholds.

LIMES ($\theta = 0.75$) necessitates approximately 30% of SILK's computation time for the Drugs experiment, it requires only roughly 5% of SILK's time for the SimCities experiments. The difference in performance is even more significant when the threshold is set higher. For example, $\theta = 0.95$ leads to LIMES necessitating only 1.6% of SILK's runtime in the SimCities experiment. The potential of our approach becomes even more obvious when one takes into consideration that we did not vary the number of exemplars in this experiment. Setting optimal values for the number of exemplars would have led to even smaller runtimes as shown by our experiments with synthetic data.

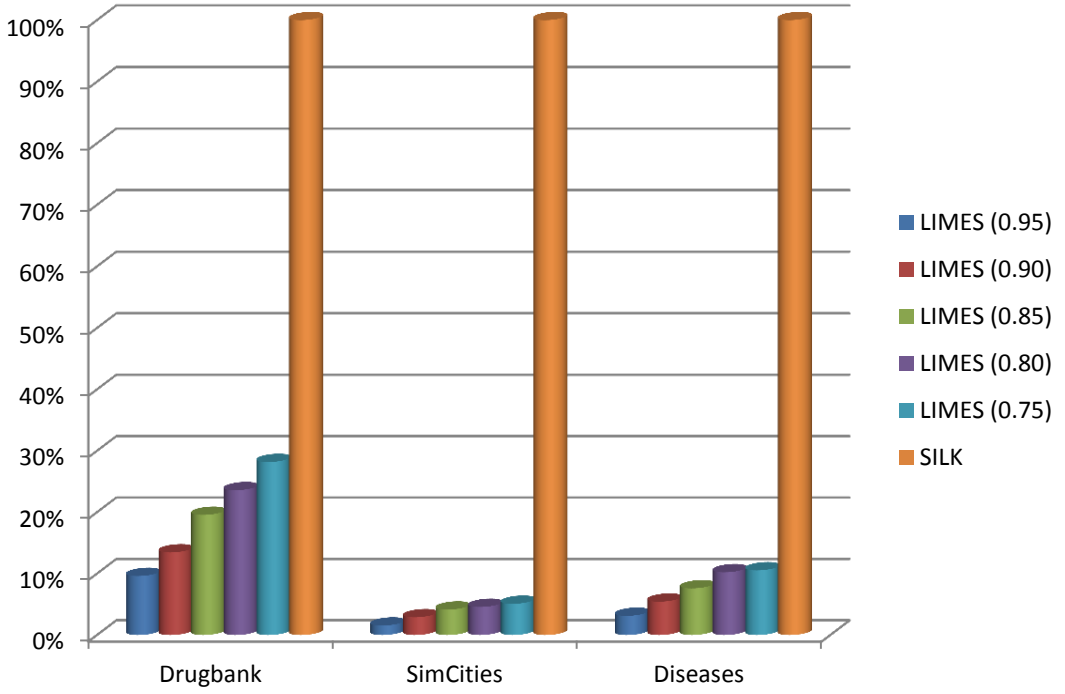


Figure 3.4: Comparison of the relative runtimes of SILK and LIMES. The number in brackets in the legend are the values of the θ threshold.

3.6 DISCUSSION AND FUTURE WORK

We presented the LIMES framework, which implements a very time-efficient approach for the discovery of links between knowledge bases on the Linked Data

Web. We evaluated our approach both with synthetic and real data and showed that it outperforms state-of-the-art approaches with respect to the number of comparisons and runtime. In particular, we showed that the speedup of our approach grows with the a-priori time complexity of the mapping task, making our framework especially suitable for handling large-scale matching tasks (cf. results of the SimCities experiment).

The main drawback of LIMES is that it is restricted to metric spaces. Thus, some popular semi-metrics such as JaroWinkler (Winkler, 1999) can not be accelerated with LIMES. To ensure that our framework can be used even with these measures, we have implemented the brute-force approach as a fall-back for comparing instances in such cases. One can easily show that our approach can be extended to semi-metrics. In future work, we will take a closer look at semi-metrics and aim at finding a relaxed triangular inequality that applies to each of them. Based on these inequalities, our framework will also use semi-metrics to compute exemplar and render link discovery based on these measures more efficient.

We also aim to explore the combination of LIMES with active learning strategies in a way, that a manual configuration of the tool becomes unnecessary.

4 THE HYPERSPHERE APPROXIMATION ALGORITHM

PREAMBLE

The subsequent chapter is based on (Ngonga Ngomo, 2011) and (Ngonga Ngomo, 2012b). These papers present the core of the processing of link specifications which underlie the default approach used by LIMES to process link specifications at the time of writing. These works were carried out by the author alone.

4.1 INTRODUCTION

The Linked Data Web has evolved from 12 knowledge bases in May 2007 to 203 knowledge bases in September 2010, i.e., in less than four years (Heath and Bizer, 2011). While the number of RDF triples available in the Linked Data Web has now surpassed 27 billion, less than 3% of these triples are links between knowledge bases (Ngonga Ngomo and Auer, 2011). Yet, links between knowledge bases play a key role in important tasks such as cross-ontology question answering (Lopez et al., 2009), large-scale inferences (Urbani et al., 2010) and data integration (Ben-David et al., 2010). Given the enormous amount of information available on the Linked Data Web, time-efficient Link Discovery (LD) frameworks have become indispensable for implementing the fourth Linked Data principle, i.e., the provision of links between data sources (Ngonga Ngomo and Auer, 2011; Volz et al., 2009). These frameworks rely on *link specifications*, which explicate conditions for computing new links between entities in knowledge bases. Due to the mere size of the Web of Data, detecting links even when using trivial specifications can be very time-demanding. Moreover, non-trivial LD tasks require complex link specifications for discovering accurate links between instances and are consequently even more challenging to optimize with respect to runtime. In this chapter, we present a novel lossless hybrid approach to LD. Our approach is based on original insights on the distribution of property domain and ranges on the Web of Data. Based on these insights, we infer the requirements to efficient LD frameworks. We then use these requirements to specify the time-efficient approaches that underlie our framework, LIMES version 0.5.[1] We show that our framework outperforms state-of-the-art frameworks by several orders of magnitude with respect to runtime without losing links.

The contributions of this chapter are as follows:

1. We present a formal grammar for link specifications that encompasses the functionality of state-of-the-art frameworks for LD.

2. Based on this grammar, we present a very time-efficient approach for LD that is based on translating complex link specifications into a combination of atomic specifications via a concatenation of operations on sets and filter operations.

1 http://limes.sf.net

3. We use this method to enable the PPJoin+ (Xiao et al., 2008) algorithm to be used for processing complex link specifications.
4. We specify and evaluate the HYpersphere aPPrOximation algorithm HYPPO, a fully novel LD approach designed to operate on numeric values.
5. We evaluate our approach against SILK (Isele et al., 2011) within three experiments and show that we outperform it by up to six orders of magnitude with respect to runtime while abiding to the constraint of not losing links.

The rest of this chapter is structured as follows: In Section 4.2, we give a brief overview of related work on LD and related research fields. Section 4.3 presents the preliminaries to our work. These preliminaries are the basis for Section 4.4, in which we specify a formal grammar for link specification and an approach to convert complex link specifications into an aggregation of atomic link specifications via set operations and filters. We subsequently present the core algorithms underlying our approach in Section 4.5. In Section 4.6, we evaluate our approaches in three different large-scale experiments and show that we outperform the state-of-the-art approach SILK. After a discussion of our findings, we present our future work and conclude.

4.2 RELATED WORK

Current frameworks for LD on the Web of Data can be subdivided into two categories: *domain-specific* and *universal* frameworks (Ngonga Ngomo and Auer, 2011). Domain-specific LD frameworks aim to discover links between knowledge bases from a particular domain. For example, the RKB knowledge base (RKB-CRS) (Glaser et al., 2009) uses Universal Resource Identifier (URI) lists to compute links between universities and conferences. Another domain-specific tool is GNAT (Raimond et al., 2008), which discovers links between music datasets by using audio fingerprinting. Further simple or domain-specific approaches can be found in (Papadakis et al., 2011; Sleeman and Finin, 2010).

Universal LD frameworks are designed to carry out mapping tasks independent from the domain of the source and target knowledge bases. For example, RDF-AI (Scharffe et al., 2009) implements a five-step approach that comprises the preprocessing, matching, fusion, interlinking and post-processing of datasets. SILK (Isele et al., 2011) (Version 2.3) implements a time-efficient and lossless approach that maps complex configurations to a multidimensional metric space. A blocking approach is then used in the metric space to reduce the number of comparisons by generating overlapping blocks. The original LIMES approach (Ngonga Ngomo and Auer, 2011) presupposes that the datasets to link are in a metric space. It then uses the triangle inequality to portion the metric space so as to compute pessimistic approximations of distances. Based on these approximations, it can discard a large number of computations without losing links.

Although LD is closely related with record linkage (Elmagarmid et al., 2007; Winkler, 2006) and deduplication (Bleiholder and Naumann, 2008), it is important to notice that LD goes beyond these two tasks as LD aims to provide the means to link entities via arbitrary relations. Different blocking techniques such as standard blocking, sorted-neighborhood, bi-gram indexing, canopy clustering and adaptive blocking have been developed by the database community to address the problem of the quadratic time complexity of brute force comparison (Köpcke et al., 2009).

In addition, very time-efficient approaches have been proposed to compute string similarities for record linkage, including AllPairs (Bayardo et al., 2007), PPJoin and PPJoin+ (Xiao et al., 2008). However, these approaches alone cannot deal with the diversity of property values found on the Web of Data as they cannot deal with numeric values. In addition, most time-efficient string matching algorithms can only deal with simple link specifications, which are mostly insufficient when computing links between large knowledge bases.

The novel version of the LIMES framework goes beyond the state of the art (including previous versions of LIMES (Ngonga Ngomo and Auer, 2011)) by integrating PPJoin+ and extending this algorithm so as to enable it to deal with complex configurations. In addition, LIMESo.5 integrates the fully novel HYPPO algorithm, which ensures that our framework can deal efficiently with numeric values and consequently with the whole diversity of data types found on the Web of Data.

4.3 PROBLEM DEFINITION

The goal of LD is to discover the set of pair of instances $(s, t) \in S \times T$ that are related by a relation R, where S and T are two not necessarily distinct sets of instances. One way to automate this discovery is to compare the $s \in S$ and $t \in T$ based on their properties using a (in general complex) similarity metric. Two entities are then considered to be linked via R if their similarity is superior to a threshold τ. We are aware that several categories of approaches can be envisaged for discovering links between instances, for example using formal inferences or semantic similarity functions. Throughout this chapter, we will consider LD via properties. This is the most common definition of instance-based LD (Ngonga Ngomo and Auer, 2011; Volz et al., 2009), which translates into the following formal specification.

Definition 2 (Link Discovery)**.** *Given two sets S (source) and T (target) of instances, a (complex) similarity measure σ over the properties of $s \in S$ and $t \in T$ and a similarity threshold $\tau \in [0, 1]$, the goal of LD is to compute the set of pairs of instances $(s, t) \in S \times T$ such that $\sigma(s, t) \geqslant \tau$.*

This problem can be expressed equivalently as follows:

Definition 3 (Link Discovery on Distances)**.** *Given two sets S and T of instances, a (complex) distance measure δ over the properties of $s \in S$ and $t \in T$and a distance threshold $\theta \in [0, \infty[$, the goal of LD is to compute the set of pairs of instances $(s, t) \in S \times T$ such that $\delta(s, t) \leqslant \theta$.*

Note that a normed similarity function σ can always be derived from a distance function δ by setting $\sigma(x, y) = (1 + \delta(x, y))^{-1}$. Furthermore, the distance threshold θ can be transformed into a similarity threshold τ by setting $\tau = (1 + \theta)^{-1}$. Consequently, distance and similarities are used interchangeably within our framework.

Although it is sometimes sufficient to define atomic similarity functions (i.e., similarity functions that operate on exactly one property pair) for LD, many LD problems demand the specification of complex similarity functions over several datatypes (numeric, strings, ...) to return accurate links. For example, while the name of bands can be used for detecting duplicate bands across different knowledge bases, linking cities from different knowledge bases requires taking more properties into consideration (e.g., the different names of the cities as well as their latitude and longitude) to compute links accurately. Consequently, linking on the Data Web demands frameworks that support complex link specifications.

4.4 LINK SPECIFICATIONS AS OPERATIONS ON SETS

In state-of-the-art LD frameworks, the condition for establishing links is usually expressed by using combinations of operations such as MAX (maximum), MIN (minimum) and linear combinations on binary similarity measures that compare property values of two instances $(s, t) \in S \times T$. Note that transformation operations may be applied to the property values (for example a lower-case transformation for strings) but do not affect our formal model. We present a formal grammar that encompasses complex link specifications as found in current LD frameworks and show how complex configurations resulting from this grammar can be translated into a sequence of set and filter operations on simple configurations. We use $\rightsquigarrow$ to denote generation rules for metrics and specifications, $\equiv$ to denote the equivalence of two specifications and $A \sqsubseteq B$ to denote that the set of links that results from specification A is a subset of the set of links that results from specification B.

Our definition of a link specification relies on the definition of *atomic similarity measures* and *similarity measures*. Generally, a similarity measure m is a function such that $m : S \times T \rightarrow [0, 1]$. We call a measure atomic (dubbed $atomicMeasure$) when it relies on exactly one similarity measure σ (e.g., trigrams similarity for strings) to compute the similarity of two instances s and t. A similarity measure m is either an atomic similarity measure $atomicMeasure$ or the combination of two similarity measures via operators OP such as MAX, MIN or linear combinations as implemented in LIMES. Thus, the following rule set for constructing metrics holds:

1. $m \rightsquigarrow atomicMeasure$
2. $m \rightsquigarrow OP(m_1, m_2)$

Note that frameworks differ in the type of operators they implement.

We call a link specification atomic ($atomicSpec$) if it compares the value of a measure m with a threshold τ, thus returning the pairs (s, t) that satisfy the condition $\sigma(s, t) \geqslant \tau$. A link specification $spec(m, \tau)$ is either an atomic link specification or the combination of two link specifications via operations such as AND (the conditions of both specifications must be satisfied, equivalent to set intersection), OR (set union), XOR (symmetric set difference), or DIFF (set difference). Thus, the following grammar for specifications holds :

1. $spec(m, \theta) \rightsquigarrow atomicSpec(m, \theta)$
2. $spec(m, \theta) \rightsquigarrow AND(spec(m_1, \theta_1), spec(m_2, \theta_2))$
3. $spec(m, \theta) \rightsquigarrow OR(spec(m_1, \theta_1), spec(m_2, \theta_2))$
4. $spec(m, \theta) \rightsquigarrow XOR(spec(m_1, \theta_1), spec(m_2, \theta_2))$
5. $spec(m, \theta) \rightsquigarrow DIFF(spec(m_1, \theta_1), spec(m_2, \theta_2))$

Most very time-efficient algorithms such as PPJoin+ operate solely on atomic measures and would not be usable if specifications could not be reduced to run only on atomic measures. For the operators MIN, MAX and linear combinations, we can reduce configurations that rely on complex measures to operations on configurations that rely on atomic measures via the following rules:

1. $spec(MAX(m_1, m_2), \theta) \equiv OR(spec(m_1, \theta), spec(m_2, \theta))$
2. $spec(MIN(m_1, m_2), \theta) \equiv AND(spec(m_1, \theta), spec(m_2, \theta))$
3. $spec(\alpha m_1 + \beta m_2, \theta) \sqsubseteq AND(spec(m_1, (\theta - \beta)/\alpha), spec(m_2, (\theta - \alpha)/\beta))$

Note that while we can derive equivalent conditions on a smaller number of dimensions for the first two operations, the simpler linking specifications that can be extracted for linear combinations are necessary to fulfill their premise, but not equivalent to the premise. Thus, in the case of linear combinations, it is important to validate the final set of candidates coming from the intersection of the two sets specified on a smaller number of dimensions against the premise by using *filters*. Given these transformations, we can reduce all complex specifications that abide by our grammar to a sequence of set and filter operations on the results of atomic measures. Consequently, we can apply very time-efficient approaches designed for atomic measures on each category of data types to process even highly complex link specifications on the Web of Data. In the following, we present the approaches used by our framework on strings and numerical values.

4.5 PROCESSING SIMPLE CONFIGURATIONS

Our framework implements a hybrid approach to LD. The first approach implemented in our framework deals exclusively with strings by harnessing the near-duplicate detection algorithm PPJoin+ (Xiao et al., 2008). Instead of mapping strings to a vector space, PPJoin+ uses a combination of three main insights to implement a very time-efficient string comparison approach. First, it uses the idea that strings with a given similarity must share a certain number of characters in their prefix to be able to have a similarity beyond the user-specified threshold. A similar intuition governs the suffix filtering implemented by PPJoin+. Finally, the algorithm makes use of the position of each word w in the index to retrieve a lower and upper bound of the index of the terms with which w might be similar. By combining these three approaches, PPJoin+ can discard a large number of non-matches. The integration of the PPJoin+ algorithm into our framework ensures that we can mitigate the pitfall of the time-demanding transformation of strings to vector spaces as implemented by multidimensional approaches. The main drawback of PPJoin+ is that it can only operate on one dimension (Köpcke et al., 2009). However, by applying the transformations of configurations specified above, we make PPJoin+ applicable to link discovery tasks with complex configurations. While mapping strings to a vector space demands some transformation steps and can be thus computationally demanding, all numeric values explicitly describe a vector space. The second approach implemented in our framework deals exclusively with numeric values and implements a novel approach dubbed HYPPO.

The HYPPO algorithm addresses the problem of efficiently mapping instance pairs $(s, t) \in S \times T$ described by using exclusively numeric values in a n-dimensional metric space. The approach assumes a distance metric δ for measuring the distance between objects and returns all pairs such that $\delta(s, t) \leqslant \theta$, where θ is a distance threshold. Let $\omega = (\omega_1, ..., \omega_n)$ and $x = (x_1, ..., x_n)$ be points in the n-dimensional space $\Omega = S \cup T$. The observation behind HYPPO is that in spaces (Ω, δ) with orthogonal, i.e., uncorrelated dimensions, distance metrics can be decomposed into the combination of functions $\phi_{i, i \in \{1...n\}}$ which operate on exactly one dimension of $\Omega : \delta = f(\phi_1, ..., \phi_n)$. For example, for Minkowsky distances of order $p > 1$,

$\phi_i(x,\omega) = |x_i - \omega_i|$ for all values of i and $\delta(x,\omega) = \sqrt[p]{\sum \phi_i(x,\omega)^p}$. Note that the Euclidean distance is the Minkowsky distance of order 2. The Minkowsky distance can be extended further by weighting the different axes of Ω. In this case, $\delta(x,\omega) = \sqrt[p]{\sum \gamma_{ii}^p \phi_i(x,\omega)^p}$ and $\phi_i(x,\omega) = \gamma_{ii}|x_i - \omega_i|$, where γ_{ii} are the entries of a positive diagonal matrix.

Some distances do exist, which do not assume an orthogonal basis for the metric space. Mahalanobis distances for example are characterized by the equation $\delta(x,\omega) = \sqrt{(x-\omega)\Gamma(x-\omega)^T}$, where Γ is a $n \times n$ covariance matrix. However, given that each space with correlated dimensions can always be transformed into an affine space with an orthonormal basis, we will assume in the remainder of this chapter that the dimensions of Ω are independent. Given this assumption, it is important to notice that the following inequality holds:

$$\phi_i(x,\omega) \leqslant \delta(x,\omega), \tag{4.1}$$

ergo, $\delta(x,\omega)$ is the upper bound of $\phi_i(x,\omega)$. Note that this is the sole condition that we pose upon δ for Hyppo to be applicable. Also note that this condition can always be brought about in a metric space by transforming its basis into an orthogonal basis.

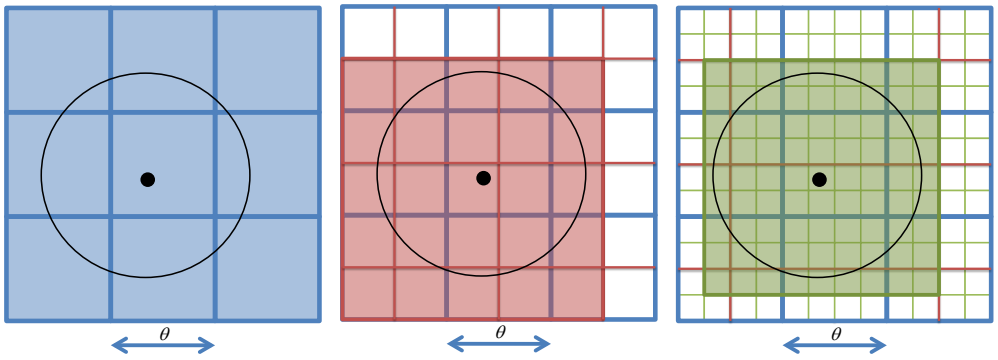
The basic intuition behind Hypo is that the hypersphere $H(\omega,\theta) = \{x \in \Omega : \delta(x,\omega) \leqslant \theta\}$ is a subset of the hypercube V defined as $V(\omega,\theta) = \{x \in \Omega : \forall i \in \{1...n\}, \phi_i(x_i,\omega_i) \leqslant \theta\}$ due to Equation 4.1. Consequently, one can reduce the number of comparisons necessary to detect all elements of $H(\omega,\theta)$ by discarding all elements which are not in $V(\omega,\theta)$ as non-matches. Hypo uses this intuition by implementing a two-step approach to LD. First, it tiles Ω into hypercubes of the same volume. Second, it compares each $s \in S$ with those $t \in T$ that lie in cubes at a distance below θ. Note that these two steps differ from the steps followed by similar algorithms (such as blocking) in two ways. First, we do not use only one but several hypercubes to approximate $H(\omega,\theta)$. Most blocking approach rely on finding *one block* that contains the elements that are to be compared with ω (Köpcke et al., 2009). In addition, Hypo is guaranteed not to lose any link, as H is completely enclosed in V, while most blocking techniques are not lossless.

Formally, let $\Delta = \theta/\alpha$. We call $\alpha \in \mathbb{N}$ the granularity parameter. Hypo first tiles Ω into the adjacent hypercubes (short: cubes) C that contain all the points ω such that $\forall i \in \{1...n\}, c_i\Delta \leqslant \omega_i < (c_i+1)\Delta$, $(c_1,...,c_n) \in \mathbb{N}^n$. We call the vector $(c_1,...,c_n)$ the coordinates of the cube C. Each point $\omega \in \Omega$ lies in the cube $C(\omega)$ with coordinates $(\lfloor \omega_i/\Delta \rfloor)_{i=1...n}$. Given such a space tiling and inequality (1), it is obvious that all elements of $H(\omega,\theta)$ lie in the set $\mathfrak{C}(\omega,\alpha)$ of cubes such that $\forall i \in \{1...n\} : |c_i - c(\omega)_i| \leqslant \alpha$. Figure 4.1 shows examples of space tilings for different values of α.

The accuracy of the approximation performed by Hypo can be computed easily: The number of cubes that approximate $H(\omega,\theta)$ is $(2\alpha+1)^n$, leading to a total volume $V_{\mathfrak{C}}(\alpha,\theta) = ((2\alpha+1)\Delta)^n = \left(\frac{2\alpha+1}{\alpha}\theta\right)^n$ that approximates $H(\omega,\theta)$. The volume $V_H(\theta)$ of $H(\omega,\theta)$ is given by $\frac{S_n\theta^n}{n}$, where S_n is the volume of a unit sphere in n dimensions, i.e., 2 for $n=1$, π for $n=2$, $\frac{4\pi}{3}$ for $n=3$ and so on. The approximation ratio

$$\frac{V_{\mathfrak{C}}(\alpha,\theta)}{V_H(\theta)} = \frac{n}{S_n}\left(\frac{2\alpha+1}{\alpha}\right)^n, \tag{4.2}$$

permits to determine the accuracy of Hypo's approximation as shown in Figure 4.2 for dimensions between 1 and 3 and values of α up to 10. Note that $V_{\mathfrak{C}}$ and



(a) $\alpha = 1$ (b) $\alpha = 2$ (c) $\alpha = 4$

Figure 4.1: Space tiling for different values of α. The coloured squares show the set of elements that must be compared with the instance located at the black dot. The points within the circle lie within the distance θ of the black dot.

V_H do not depend on ω and that $\frac{V_{\mathfrak{C}}(\alpha,\theta)}{V_H(\theta)}$ does not depend on θ. Furthermore, note that the higher the value of α, the better the accuracy of HYPPO. Yet, higher values of α also lead to an exponentially growing number of hypercubes $|\mathfrak{C}(\omega, \alpha)|$ and thus to longer runtimes when constructing $\mathfrak{C}(\omega, \alpha)$ to approximate $H(\omega, \theta)$. Once the space tiling has been completed, all that remains to do is to compare each $s \in S$ with all the $t \in T \cap (\bigcup C \in \mathfrak{C}(\omega, \alpha))$ and to return those pairs of entities such that $\delta(s, t) \leqslant \theta$. Algorithm 4.1 shows HYPPO's pseudocode.

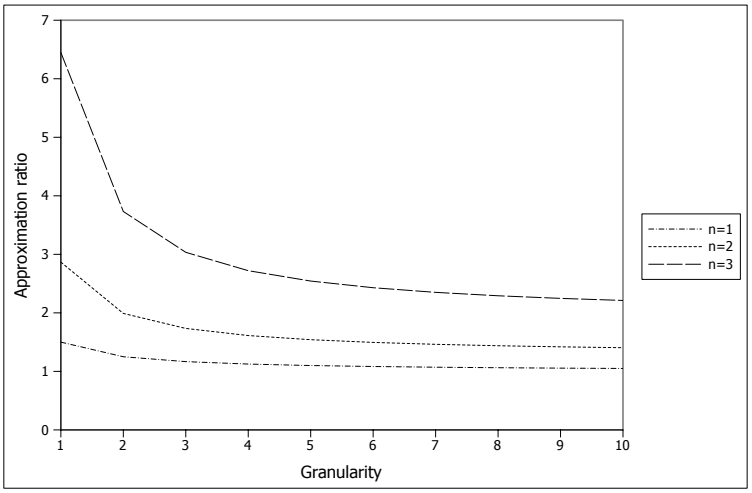


Figure 4.2: Approximation ratio for $n \in \{1, 2, 3\}$. The x-axis shows values of α while the y-axis shows the approximation ratios.

4.6 EVALUATION

4.6.1 *Experiments with* HYPPO

In our first series of experiments, we evaluated HYPPO within six hypothetical use cases and compared it with SILK. We ran all experiments with θ- and α-values between 1 and 16. All datasets were retrieved from DBpedia, as it contains a large

Algorithm 4.1 Current implementation of HYPPO

Require: S, T, θ, δ, α as defined above
1: Mapping $M := \emptyset$
2: $\Delta = \theta/\alpha$
3: **for** $\omega \in S \cup T$ **do**
4: $\quad C(\lfloor \omega_1/\Delta \rfloor, ..., \lfloor \omega_n/\Delta \rfloor) := C(\lfloor \omega_1/\Delta \rfloor, ..., \lfloor \omega_n/\Delta \rfloor) \cup \{\omega\}$
5: **end for**
6: **for** $s \in S$ **do**
7: $\quad$ **for** $C \in \mathfrak{C}(s, \alpha)$ **do**
8: $\quad\quad$ **for** $t \in C \cap T$ **do**
9: $\quad\quad\quad$ **if** $\delta(s,t) \leqslant \theta$ **then**
10: $\quad\quad\quad\quad M := M \cup (s,t)$
11: $\quad\quad\quad$ **end if**
12: $\quad\quad$ **end for**
13: $\quad$ **end for**
14: **end for**
15: **return** M

Experiment	# Instances	A-priori Complexity	# Dimensions
Town	27,525	758×10^6	1
Books	14,714	217×10^6	1
Vacations	21,925	481×10^6	2
Actors	15,909	253×10^6	2
Series*	4,841	23×10^6	3
Hydrology	19,095	365×10^6	3

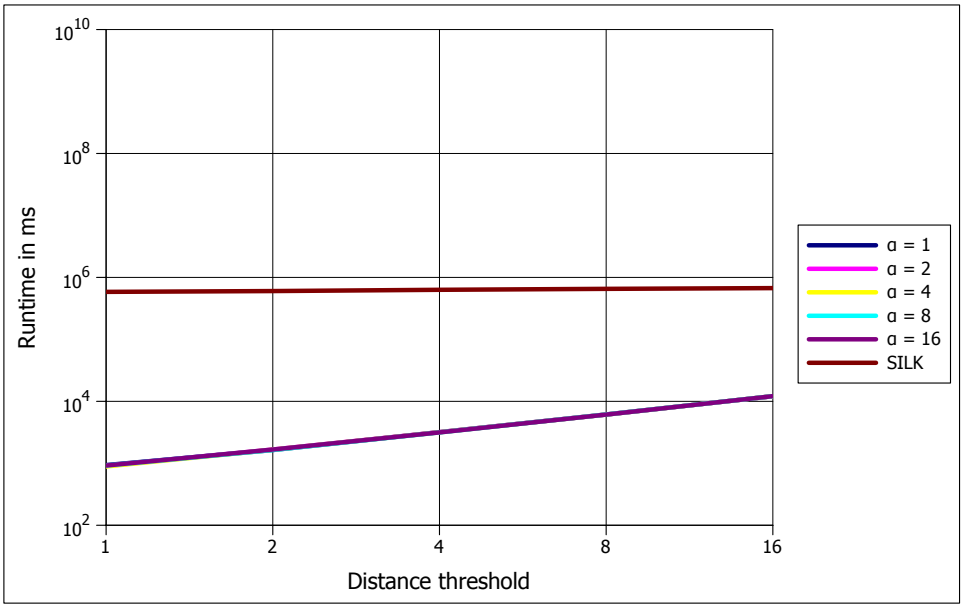
Table 4.1: Summary of experimental setups for experiments on HYPPO

amount of general knowledge. In all experiments, we used the normed similarity based on the Euclidean Distance.

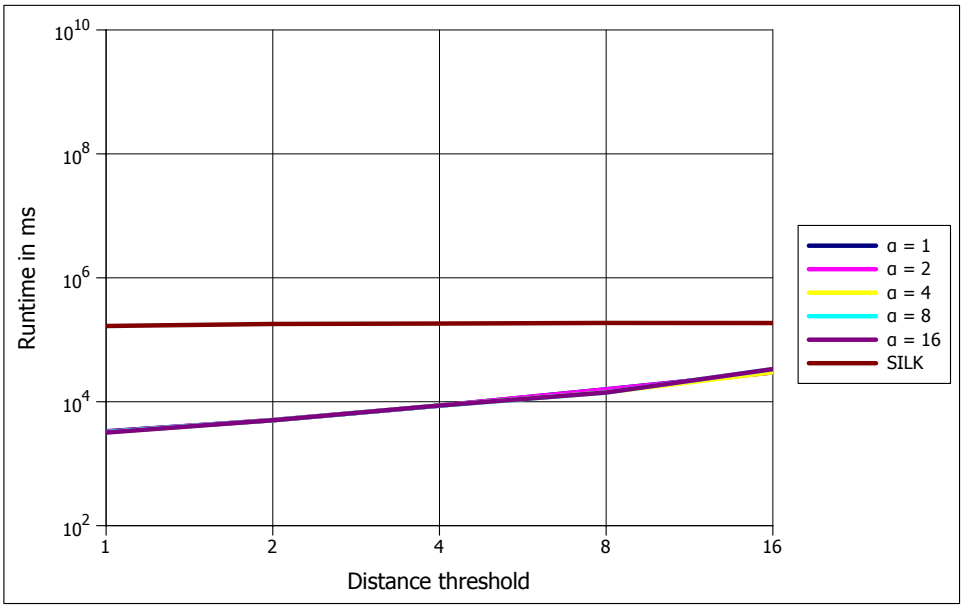
The goal of the first experiment, dubbed *Towns*, was to link towns with similar populations for demographic studies. The second experiment, dubbed *Books*, was also one-dimensional and aimed at finding books with a similar number of pages for a book recommendation system. The third and fourth experiment were two-dimensional. *Vacations*, the third experiment, aimed to detect similar vacation sites by comparing their population and elevation for a tourism portal. The fourth experiment dubbed *Actors* linked people with similar height and weight with the goal of retrieving possible candidates for acting given parts in movies. The fifth and sixth experiments were carried out on three dimensions. The fifth experiment was designed to link television *Series* with similar number of episodes, number of seasons and runtime for a movie portal. Finally, the sixth experiment links city with the same elevation, land area and water area for studies on *Hydrology*. An overview of the experiments in given in Table 4.1.

The results of this series of experiments are shown in Figure 4.3, Figure 4.4 and Figure 4.5. We outperform SILK by more than five orders of magnitude, e.g., in the Series (see Figure 4.5a) experiments. Note that the experiments *Series* was terminated after 2 days of runtime by SILK. The runtime of 48 hours reported

in the experiment is thus strictly inferior to the real runtime of SILK. Surprisingly, the results of these series of experiments suggest that the runtimes of HYPPO barely depend on the value of α. We expected the runtimes to decrease significantly up to a certain value of α and then to increase or remain constant. Augmenting α did lead to other runtimes, yet the difference was minute, i.e., around 5% at most. As the results of our large-scale experiments show, these minute differences were solely due to the small size of experiments. In large-scale experiments, α can have a considerable influence on the total runtime.

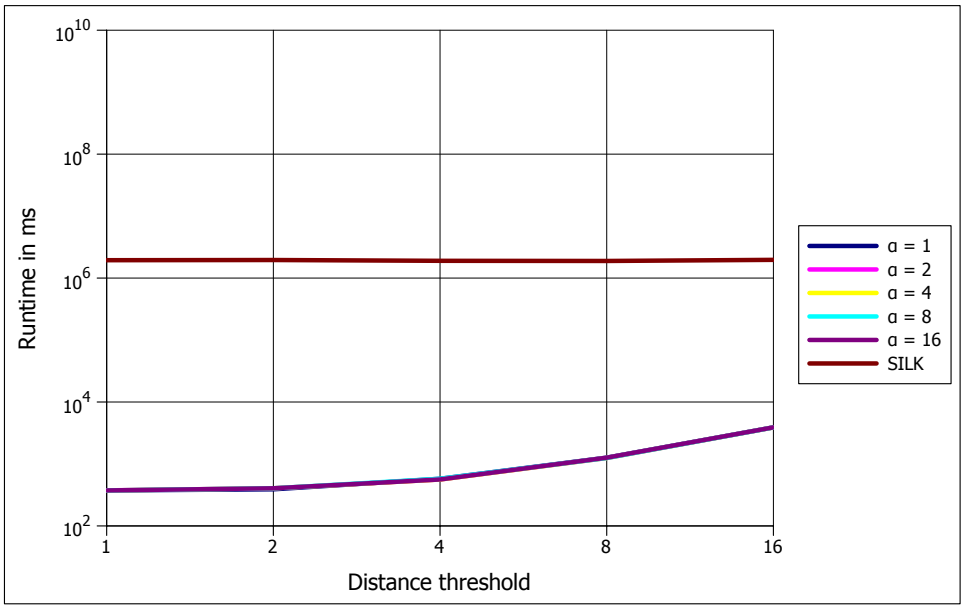


(a) Towns (n=1)

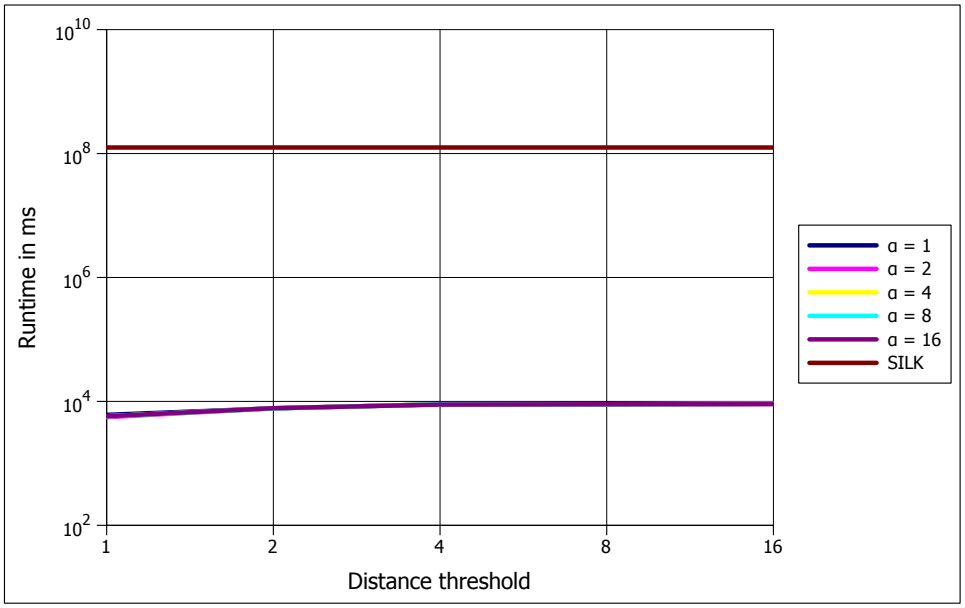


(b) Books (n=1)

Figure 4.3: Comparison of HYPPO and SILK for experiments of dimension 1. The x-axis shows $\log_2(\theta)$ while the y-axis shows the runtime in ms on a logarithmic scale. Note that given the size of the experiment, the limes for different values of α are superimposed.



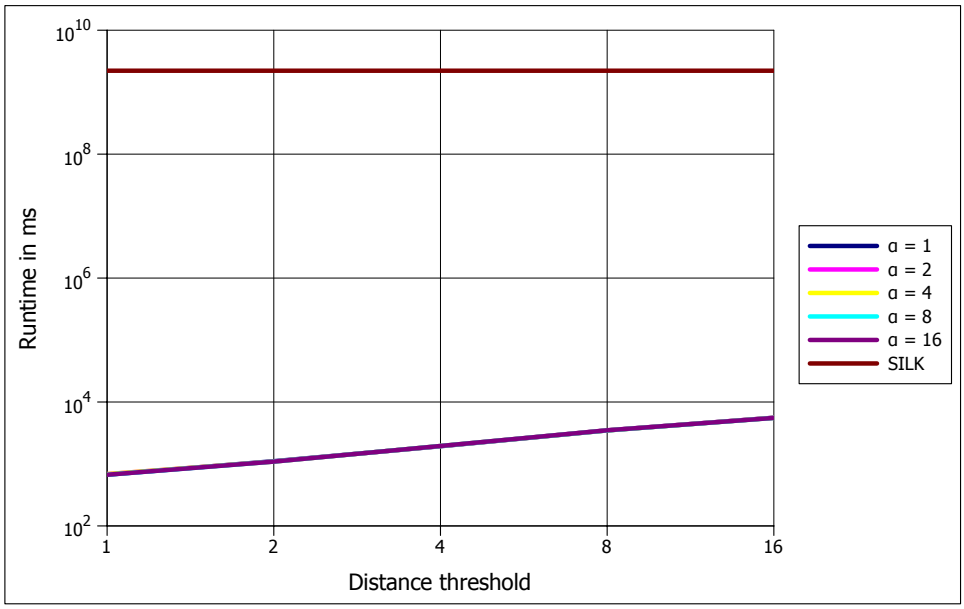
(a) Vacations (n=2)



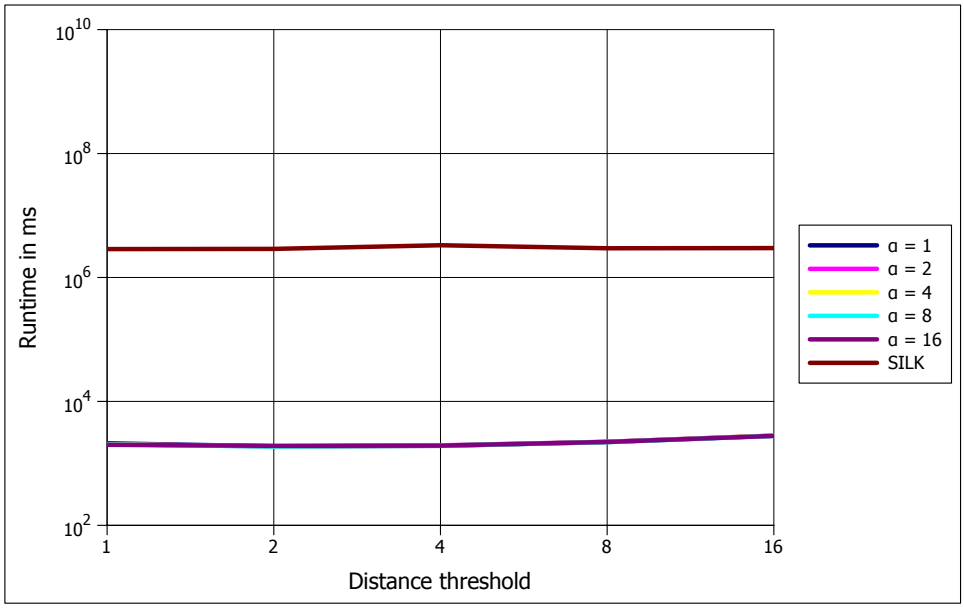
(b) Actors (n=2)

Figure 4.4: Comparison of HYPPO and SILK for experiments of dimension 2. The x-axis shows $\log_2(\theta)$ while the y-axis shows the runtime in ms on a logarithmic scale. Note that given the size of the experiment, the limes for different values of α are superimposed.

Note that we aimed initially to also consider large problems in this series of experiments (i.e., problems with a complexity above 10^9). Yet, our seventh series of experiments, dubbed *Places*, forced us to limit ourselves to average-sized experiments. The goal of Places was to detect which ones of the 209,630 instances (a-priori complexity of approximately 44×10^9) of `dbpedia-owl:Place` have similar populations. While the experiment led to a runtime of 95,946ms with LIMES for $\theta = 1$ and $\alpha = 1$, we had to terminate the computation with SILK after 7 days (i.e., 0.6×10^9 ms), thus more than 7200 times LIMES' runtime.



(a) Series* (n=3)



(b) Hydrology (n=3)

Figure 4.5: Comparison of HYPPO and SILK for experiments of dimension 3. The x-axis shows $\log_2(\theta)$ while the y-axis shows the runtime in ms on a logarithmic scale. Note that given the size of the experiment, the limes for different values of α are superimposed.

4.6.2 *Experiments with LIMES*

We compared our framework (i.e., LIMES Version 0.5) with SILK version 2.3. in three large-scale experiments of different complexity based on geographic data. We chose SILK because (to the best of our knowledge) it is the only other LD framework that allows the specification of such complex linking experiments. We ran all experiments on the same computer running a Windows 7 Enterprise 64-bit installation on a 2.8GHz i7 processor with 8GB RAM. The JVM was allocated 7.4GB

Experiment	$\|S\| \times \|T\|$	Dims	Source	Target	Thresholds
Villages*	26,717 × 103,175	2	DBpedia	LGD	τ_s, θ_p
Cities*	36,877 × 39,800	3	Geonames	DBpedia	τ_s, θ_p
Geo-Locations*	50,031 × 74,458	4	LGD	GeoNames	τ_s, θ_p, θ_l

Table 4.2: Summary of experimental setups for LIMES and SILK. Dims stands for dimensions.

RAM. For each tool we measured exclusively the time needed for computing the links. All experiments were carried out 5 times except when stated otherwise. In all cases, we report best runtimes. Experiments marked with an asterisk would have lasted longer than 48 hours when using SILK and were not computed completely. Instead, SILK's runtime was approximated by extrapolating the time needed by the software to compute 0.1% of the links.

We chose to use geographic datasets because they are large and allow the use of several attributes for linking. In the first experiment, we computed links between *villages* in DBpedia and LinkedGeoData based on the `rdfs:label` and the `population` of instances. The link condition was twofold: (1) the difference in population had to be lower or equal to θ and (2) the labels had to have a trigram similarity larger or equal to τ. In the second experiment, we aimed to link towns and *cities* from DBpedia with populated places in Geonames. We used the names (`gn:name`), alternate names (`gn:alternateName`) and `population` of cities as criteria for the comparison. Finally, we computed links between *Geo-locations* in LinkedGeoData and GeoNames by using 4 combinations of criteria for comparing entities: their longitude (`wgs84:long`), latitude (`wgs84:lat`), preferred names and names.

The setup of the experiments is summarized in Table 4.2. We used two threshold setups. In the *strict setup*, the similarity threshold τ_s on strings was set to 0.9, the maximal difference in population θ_p was set to 9 and the maximal difference in latitude and longitude θ_l was set to 1. In the *lenient setup*, τ_s was set to 0.7 and θ_p to 19. The lenient setup was not used in the Geo-Locations experiments because it led to too many links, which filled up the 7.4G of RAM allocated to both tools and led to swapping, thus falsifying the evaluation of the runtimes. In all setups, we use the trigrams similarity metrics for strings and the euclidean distance for numeric values.

Our results (see Figure 4.6) confirm that we outperform SILK by several orders of magnitude in all setups. In the Cities experiment, we are more than 6 orders of magnitude faster than SILK. We compared the runtimes of LIMES for different values of α as shown in Figure 4.7. Our results show that our assumption on the relation between α and runtimes is accurate as finding the right value for α can reduce the total runtime of the algorithm by approximately 40% (see Geo-Locations, $\alpha = 4$). In general, setting α to values between 2 and 4 leads to an improved performance in all experiments.

4.7 DISCUSSION AND FUTURE WORK

In this chapter, we introduced and evaluated a novel hybrid approach to LD. We presented original insights on the conversion of complex link specifications into

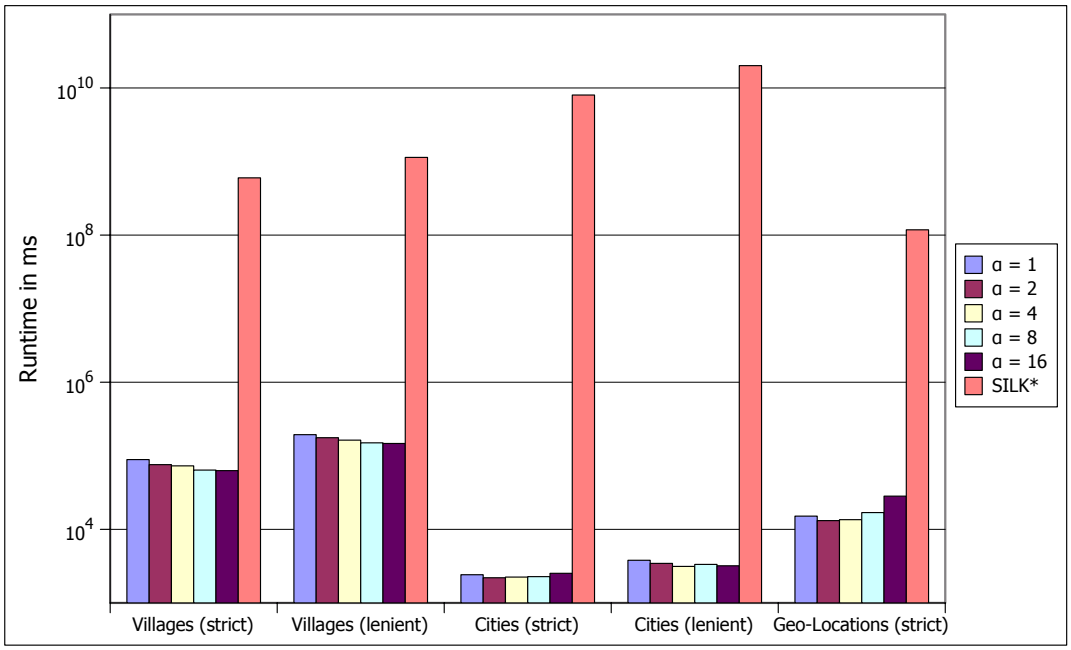


Figure 4.6: Comparison of the runtime of LIMES and SILK on large-scale link discovery tasks

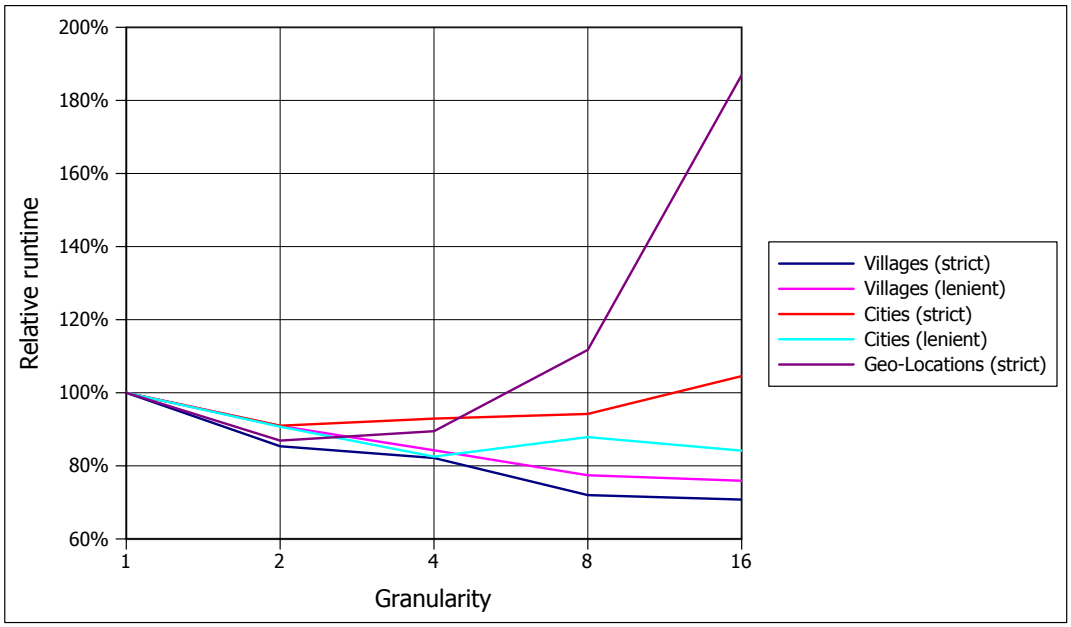


Figure 4.7: Runtimes of LIMES relatively to the runtime for $\alpha = 1$

simple link specifications. Based on these conversions, we inferred that efficient means for processing simple link specifications are the key for time-efficient linking. We then presented the two time-efficient approaches implemented in LIMES0.5 and showed how these approaches can be combined for time-efficient linking. A thorough evaluation of our framework in three large-scale experiments showed that we outperform SILK by more than 6 orders of magnitude while not losing a single link.

One of the central innovations of this chapter is the HYpersphere aPPrOximation algorithm HYPPO. Although it was defined for numeric values, HYPPO can be easily generalized to the efficient computation of the similarity of pairs of entities that are totally ordered, i.e., to all sets of entities $e = (e_1, ..., e_n) \in E$ such that a real function f_i exists, which preserves the order $\succ$ on the i^{th} dimension of E, ergo $\forall e, e' \in E : e_i \succ e'_i \rightarrow f(e_i) > f(e'_i)$. Yet, it is important to notice that such a function can be very complex and thus lead to overheads that may nullify the time gain of HYPPO. In future work, we will aim to find such functions for different data types. In addition, we will aim to formulate an approach for determining the best value of α for any given link specification. The new version of LIMES promises to be a stepping stone for the creation of a multitude of novel semantic applications, as it is time-efficient enough to make complex interactive scenarios for link discovery possible even at large scale.

5 REDUCTION-RATIO-OPTIMAL LINK DISCOVERY

PREAMBLE

This chapter is based on (Ngonga Ngomo, 2014). This paper build upon the HYPPO algorithm as presented in the previous chapter and addresses (for the first time) the optimization of the reduction ratio of link discovery algorithms. This work was carried out by the author alone.

5.1 INTRODUCTION

One of the key principles of the Linked Data paradigm is the inclusion of links between datasets (Auer et al., 2013a; Ngonga Ngomo et al., 2014). While this principle is central for tasks such as federated querying (Schwarte et al., 2011), cross-ontology question answering (Lopez et al., 2009), large-scale inferences (Urbani et al., 2010) and data integration (Ben-David et al., 2010), it is increasingly tedious to implement manually. One of the main difficulty behind the discovery of links is its intrinsic time complexity. Over the last five years, the Linked Data Web has evolved from 12 knowledge bases (May 2007) to more than 295 knowledge bases in September 2011 which contain more than 31 billion triples.[1] The combination of the mere size of these knowledge bases and the quadratic a-priori time complexity of Link Discovery leads to brute-force algorithms requiring weeks and even longer to compute links between large knowledge bases such as DBpedia[2] and LinkedGeoData.[3] Addressing this challenge demands the development of time-efficient and lossless solutions for the computation of links. Link Discovery frameworks such as LIMES (Ngonga Ngomo and Auer, 2011; Ngonga Ngomo, 2011) and SILK (Isele et al., 2011) have been designed to address this challenge. Yet, none of the manifold approaches they implement provides theoretical guarantees with respect to their performance. Thus, so far, it was impossible to predict how Link Discovery frameworks would perform w.r.t. time or space requirements. Consequently, the deployment of techniques such as customized memory management (Botelho and Ziviani, 2007) or time-optimization strategies (Vaquero et al., 2011) (e.g., automated scaling for cloud computing when provided with very complex linking tasks) was rendered very demanding if not impossible.

In this chapter, we introduce the novel approach $\mathcal{HR}^3$. Similar to the HYPPO algorithm (Ngonga Ngomo, 2011) (on whose formalism it is based), $\mathcal{HR}^3$ assumes that the property values that are to be compared are expressed in an affine space with a Minkowski distance. Consequently, it can be most naturally used to process the portion of link specifications that compare numeric values (e.g., temperatures, elevations, populations, etc.). $\mathcal{HR}^3$ goes beyond the state of the art by being able to carry out Link Discovery tasks with *any achievable reduction ratio* (Elfeky et al., 2002). This theoretical guarantee is of practical importance, as it does not only

1 http://www4.wiwiss.fu-berlin.de/lodcloud/state/, accessed January 2013.
2 http://dbpedia.org, accessed January 2013.
3 http://linkedgeodata.org, accessed January 2013.

allow our approach to be more time-efficient than the state of the art but also lays the foundation for the implementation of customized memory management and time-optimization strategies for Link Discovery. The three main contributions of this chapter are thus as follows:

1. We present a novel indexing scheme for hypercubes in metric spaces with Minkowski distances. This scheme builds the basis upon which $\mathcal{HR}^3$ discards unnecessary comparisons.
2. We prove formally that $\mathcal{HR}^3$'s reduction ratio can be made arbitrarily close to the optimal reduction ratio. For this purpose, we first define the relative reduction ratio (RRR). We then show that $\mathcal{HR}^3$'s RRR converges towards a lower bound and prove this bound to be exactly 1.
3. We show experimentally that in addition to providing theoretical guarantees, our approach outperforms the state of the art. For this purpose, we compare the number of comparisons carried out by $\mathcal{HR}^3$ and HYPPO. In addition, we compare $\mathcal{HR}^3$'s runtime with that of HYPPO (as implemented in LIMES) and SILK[4].

The rest of this chapter is structured as follows: In Section 5.2, we present preliminaries and the notation used to formalize our approach $\mathcal{HR}^3$. We also introduce the relative reduction ratio RRR. We then prove that our algorithm can achieve any RRR score larger than 1 and that we can therewith achieve any possible reduction ratio (Section 5.3). After a short presentation of the implementation of our algorithm in Section 5.4, we evaluate our approach against SILK and HYPPO in four experiments in Section 5.5. Subsequently, in Section 5.6, we give an overview of previous approaches to Link Discovery. Finally, we discuss our findings and conclude in Section 5.7.

5.2 PRELIMINARIES

In this section, we present the preliminaries necessary to understand the subsequent parts of this work. In particular, we define the problem of Link Discovery, the reduction ratio and the relative reduction ratio formally as well as give an overview of space tiling for Link Discovery. The subsequent description of $\mathcal{HR}^3$ relies partly on the notation presented in this section.

5.2.1 *Link Discovery*

The goal of Link Discovery is to compute the set of pair of instances $(s, t) \in S \times T$ that are related by a relation R, where S and T are two not necessarily distinct sets of instances. One way to automate this discovery is to compare the $s \in S$ and $t \in T$ based on their properties using a distance measure. Two entities are then considered to be linked via R if their distance is less or equal to a threshold θ (Ngonga Ngomo and Auer, 2011).

Definition 4 (Link Discovery on Distances)**.** *Given two sets* S *and* T *of instances, a distance measure* δ *over the properties of* $s \in S$ *and* $t \in T$*and a distance threshold*

4 The algorithm was implemented in the new version of LIMES, of which a demo is available at http://limes.aksw.org.

$\theta \in [0, \infty[$, *the goal of Link Discovery is to compute the set* $M = \{(s, t, \delta(s, t)) : s \in S \wedge t \in T \wedge \delta(s, t) \leqslant \theta\}$.

Note that in this thesis, we are only interested in lossless solutions, i.e., solutions that are able to find all pairs that abide by the definition given above.

5.2.2 *Reduction Ratio*

A brute-force approach to Link Discovery would execute a Link Discovery task on S and T by carrying out $|S||T|$ comparisons. One of the key ideas behind time-efficient Link Discovery algorithms $\mathcal{A}$ is to reduce the number of comparisons that are effectively carried out to a number $C(\mathcal{A}) < |S||T|$ (Song and Heflin, 2011). The reduction ratio RR of an algorithm $\mathcal{A}$ is given by

$$RR(\mathcal{A}) = 1 - \frac{C(\mathcal{A})}{|S||T|}. \tag{5.1}$$

$RR(\mathcal{A})$ captures how much of the Cartesian product $|S||T|$ was not explored before the output of $\mathcal{A}$ was reached. It is obvious that even an optimal lossless solution which performs only the necessary comparisons cannot achieve a RR of 1. Let C_{min} be the minimal number of comparisons necessary to complete the Link Discovery task without losing recall, i.e., $C_{min} = |M|$. We define the relative reduction ratio $RRR(\mathcal{A})$ as the proportion of the minimal number of comparisons that was carried out by the algorithm $\mathcal{A}$ before it terminated. Formally

$$RRR(\mathcal{A}) = \frac{1 - \frac{C_{min}}{|S||T|}}{1 - \frac{C(\mathcal{A})}{|S||T|}} = \frac{|S||T| - C_{min}}{|S||T| - C(\mathcal{A})}. \tag{5.2}$$

$RRR(\mathcal{A})$ indicates how close $\mathcal{A}$ is to the optimal solution with respect to the number of candidates it tests. Given that $C(\mathcal{A}) \geqslant C_{min}$, $RRR(\mathcal{A}) \geqslant 1$. Note that the larger the value of $RRR(\mathcal{A})$, the poorer the performance of $\mathcal{A}$ with respect to the task at hand.

The main observation that led to this work is that while most algorithms aim to optimize their RR (and consequently their RRR), current approaches to Link Discovery do not provide any guarantee with respect to the RR (and consequently the RRR) that they can achieve. In this work, we present an approach to Link Discovery in metric spaces whose RRR is guaranteed to converge to 1.

5.2.3 *Space Tiling for Link Discovery*

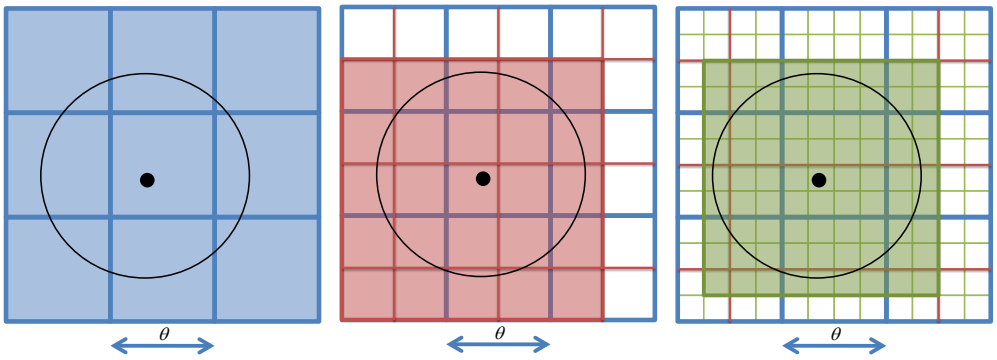
Our approach, $\mathcal{HR}^3$, builds upon the same formalism on which the HYPPO algorithm relies, i.e., space tiling. HYPPO addresses the problem of efficiently mapping instance pairs $(s, t) \in S \times T$ described by using exclusively numeric values in a n-dimensional metric space and has been shown to outperform the state of the art in previous work (Ngonga Ngomo, 2011). The observation behind space tiling is that in spaces (Ω, δ) with orthogonal, (i.e., uncorrelated) dimensions,[5] common metrics for Link Discovery can be decomposed into the combination of functions $\phi_{i, i \in \{1 \ldots n\}}$ which operate on exactly one dimension of $\Omega : \delta = f(\phi_1, ..., \phi_n)$.

5 Note that in all cases, a space transformation exists that can map a space with correlated dimensions to a space with uncorrelated dimensions.

For Minkowski distances of order p, $\phi_i(x,\omega) = |x_i - \omega_i|$ for all values of i and $\delta(x,\omega) = \sqrt[p]{\sum_{i=1}^{n} \phi_i^p(x,\omega)^p}$. A direct consequence of this observation is the inequality $\phi_i(x,\omega) \leqslant \delta(x,\omega)$. The basic insight that results this observation is that the hypersphere $H(\omega,\theta) = \{x \in \Omega : \delta(x,\omega) \leqslant \theta\}$ is a subset of the hypercube V defined as $V(\omega,\theta) = \{x \in \Omega : \forall i \in \{1...n\}, \phi_i(x_i,\omega_i) \leqslant \theta\}$. Consequently, one can reduce the number of comparisons necessary to detect all elements of $H(\omega,\theta)$ by discarding all elements which are not in $V(\omega,\theta)$ as non-matches. Let $\Delta = \theta/\alpha$, where $\alpha \in \mathbb{N}$ is the *granularity parameter* that controls how fine-grained the space tiling should be (see Figure 5.1 for an example). We first tile Ω into the adjacent hypercubes (short: cubes) C that contain all the points ω such that

$$\forall i \in \{1...n\}, c_i\Delta \leqslant \omega_i < (c_i+1)\Delta \text{ with } (c_1,...,c_n) \in \mathbb{N}^n. \tag{5.3}$$

We call the vector $(c_1,...,c_n)$ the coordinates of the cube C. Each point $\omega \in \Omega$ lies in the cube $C(\omega)$ with coordinates $(\lfloor \omega_i/\Delta \rfloor)_{i=1...n}$. Given such a space tiling, it is obvious that $V(\omega,\theta)$ consists of the union of the cubes such that $\forall i \in \{1...n\} : |c_i - c(\omega)_i| \leqslant \alpha$.



(a) $\alpha = 1$ (b) $\alpha = 2$ (c) $\alpha = 4$

Figure 5.1: Space tiling for different values of α. The coloured squares show the set of elements that must be compared with the instance located at the black dot. The points within the circle lie within the distance θ of the black dot. Note that higher values of α lead to a better approximation of the hypersphere but also to more hypercubes.

Like most of the current algorithms for Link Discovery, space tiling does not provide optimal performance guarantees. The main goal of this chapter is to build upon the tiling idea so as to develop an algorithm that can achieve any possible RR. In the following, we present such an algorithm, $\mathcal{HR}^3$.

5.3 APPROACH

The goal of the $\mathcal{HR}^3$ algorithm is to efficiently map instance pairs $(s,t) \in S \times T$ that are described by using exclusively numeric values in a n-dimensional metric space where the distances are measured by using any Minkowski distance of order $p \geqslant 2$. To achieve this goal, $\mathcal{HR}^3$ relies on a *novel indexing scheme* that allows achieving any RRR greater than or equal to than 1. In the following, we first present our new indexing scheme and show that we can discard more hypercubes than simple

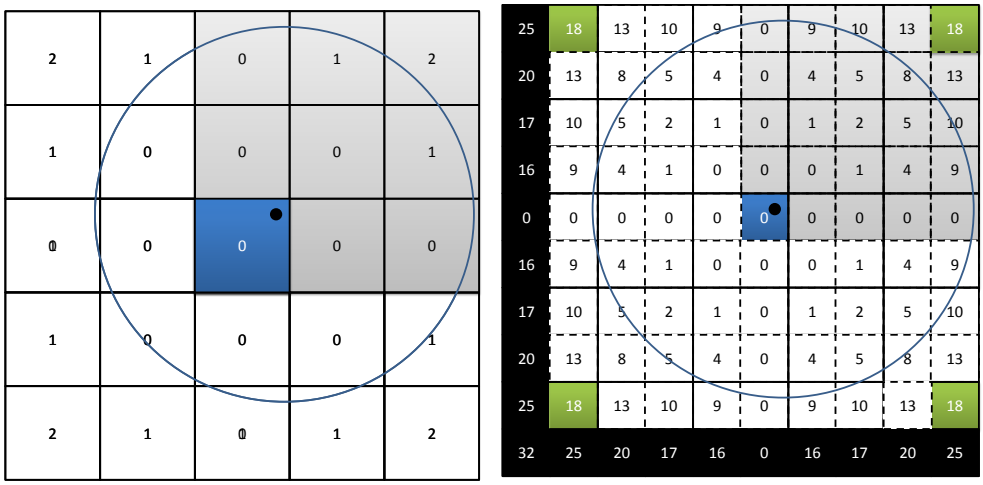
space tiling for all granularities α such that $n(\alpha-1)^p > \alpha^p$. We then prove that by these means, our approach can achieve any RRR greater than 1, therewith proving the *optimality of our indexing scheme* with respect to RRR.

5.3.1 *Indexing scheme*

Let $\omega \in \Omega = S \cup T$ be an arbitrary reference point. Furthermore, let δ be the Minkowski distance of order p. We define the $index$ function as follows:

$$index(C,\omega) = \begin{cases} 0 \text{ if } \exists i : |c_i - c(\omega)_i| \leqslant 1 \text{ with } i \in \{1,\dots,n\}, \\ \sum_{i=1}^{n} (|c_i - c(\omega)_i| - 1)^p \text{ else,} \end{cases} \tag{5.4}$$

where C is a hypercube resulting from a space tiling and $\omega \in \Omega$. Figure 5.2 shows an example of such indexes for $p = 2$ with $\alpha = 2$ (Figure 5.2a) and $\alpha = 4$ (Figure 5.2b).



(a) $\alpha = 2$ (b) $\alpha = 4$

Figure 5.2: Space tiling and resulting index for a two-dimensional example. Note that the index in both subfigures was generated for exactly the same portion of space. The black dot stands for the position of ω.

Note that the blue square with index 0 contains the reference point ω. Also note that our indexing scheme is symmetric with respect to $C(\omega)$. Thus, it is sufficient to prove the subsequent lemmas for hypercubes C such that $c_i > c(\omega)_i$. In Figure 5.2, this is the upper right portion of the indexed space with the gray background. Finally, note that the maximal index that a hypercube can achieve is $n(\alpha-1)^p$ as $\max|c_i - c_i(\omega)| = \alpha$ per construction of $H(\omega,\theta)$.

The indexing scheme proposed above guarantees the following:

Lemma 5.1. $index(C,\omega) = x \rightarrow \forall s \in C(\omega)\ \forall t \in C\ \delta^p(s,t) > x\Delta^p$.

Proof. This lemma is a direct implication of the construction of the index. $index(C,\omega) = x$ implies that

$$\sum_{i=1}^{n} (c_i - c(\omega)_i - 1)^p = x.$$

Now given the definition of the coordinates of a cube (Equation 5.3), the following holds:

$$\forall s \in C(\omega)\ \forall t \in C\ |s_i - t_i| \geqslant (|c_i - c(\omega)_i| - 1)\Delta.$$

Consequently,

$$\forall s \in C(\omega)\ \forall t \in C\ \sum_{i=1}^{n} |s_i - t_i|^p \geqslant \sum_{i=1}^{n} (|c_i - c(\omega)_i| - 1)^p \Delta^p.$$

By applying the definition of the Minkowski distance of the index function, we finally get $\forall s \in C(\omega)\ \forall t \in C\ \delta^p(s,t) > x\Delta^p$. □

Note that given that $\omega \in C(\omega)$, the following also holds:

$$\mathrm{index}(C,\omega) = x \rightarrow \forall t \in C : \delta^p(\omega,t) > x\Delta^p. \tag{5.5}$$

5.3.2 $\mathcal{HR}^3$

The main insight behind $\mathcal{HR}^3$ is that in spaces with Minkowski distances, the indexing scheme proposed above allows to safely (i.e., without dismissing correct matches) discard more hypercubes than when using simple space tiling. More specifically,

Lemma 5.2. $\forall s \in S : \mathrm{index}(C,s) > \alpha^p$ *implies that all* $t \in C$ *are non-matches.*

Proof. This lemma follows directly from Lemma 5.1 as

$$\mathrm{index}(C,s) > \alpha^p \rightarrow \forall t \in C, \delta^p(s,t) > \Delta^p \alpha^p = \theta^p. \tag{5.6}$$

□

For the purpose of illustration, let us consider the example of $\alpha = 4$ and $p = 2$ in the two-dimensional case displayed in Figure 5.2b. Lemma 5.2 implies that any point contained in a hypercube C_{18} with index 18 cannot contain any element t such that $\delta(s,t) \leqslant \theta$. While space tiling would discard all black cubes in Figure 5.2b but include the elements of C_{18} as candidates, $\mathcal{HR}^3$ discards them and still computes exactly the same results, yet with a better (i.e., smaller) RRR.

One of the direct consequences of Lemma 5.2 is that $n(\alpha-1)^p > \alpha^p$ is a necessary and sufficient condition for $\mathcal{HR}^3$ to achieve a better RRR than simple space tiling. This is simply due to the fact that the largest index that can be assigned to a hypercube is $\sum_{i=1}^{n} (\alpha-1)^p = n(\alpha-1)^p$. Now, if $n(\alpha-1)^p > \alpha^p$, then this cube can be discarded. For $p = 2$ and $n = 2$ for example, this condition is satisfied for $\alpha \geqslant 4$. Knowing this inequality is of great importance when deciding on when to use $\mathcal{HR}^3$ as discussed in Section 5.5.

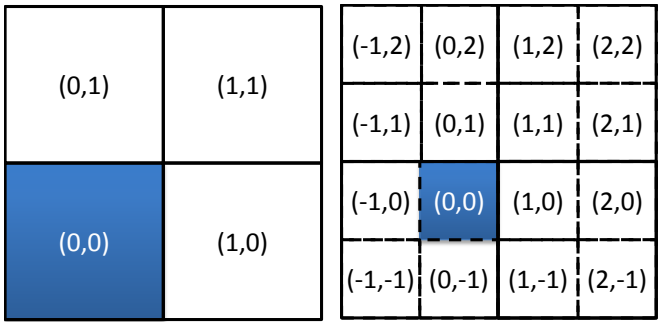
Let $\mathcal{H}(\alpha,\omega) = \{C : \mathrm{index}(C,\omega) \leqslant \alpha^p\}$. $\mathcal{H}(\alpha,\omega)$ is the approximation of the hypersphere $H(\omega) = \{\omega' : \delta(\omega,\omega') \leqslant \theta\}$ generated by $\mathcal{HR}^3$. We define the volume of $\mathcal{H}(\alpha,\omega)$ as

$$V(\mathcal{H}(\alpha,\omega)) = |\mathcal{H}(\alpha,\omega)|\Delta^p. \tag{5.7}$$

To show that given any $r > 1$, the approximation $\mathcal{H}(\alpha,\omega)$ can always achieve a an $\mathrm{RRR}(\mathcal{HR}^3) \leqslant r$, we begin by showing that

Lemma 5.3. $\forall \alpha > 1\ V(\mathcal{H}(\alpha, \omega)) > V(\mathcal{H}(2\alpha, \omega))$.

Proof. Any cube C discarded by $\mathcal{HR}^3(\alpha)$ is split into 2^n cubes $\mathcal{C}$ by $\mathcal{HR}^3(2\alpha)$, each of which has the coordinates $2c_i$ or $2c_i + 1$. In the worst case for $\mathcal{HR}^3$, ω is assigned the coordinates $2c_i(\omega) + 1$. Figure 5.3 exemplifies this property of our indexing scheme. Figure 5.3b is an indexing of the same space with the the twofold granularity.



(a) Initial coordinates (b) Coordinates with two-fold granularity

Figure 5.3: Comparison of coordinates for granularities α and 2α

When processed by $\mathcal{HR}^3(2\alpha)$, the minimal index of a hypercube $\mathcal{C}$ is then given by

$$\min index(\mathcal{C}) = \sum_{i=1}^{n} (2c_i - (2c_i(\omega) + 1) - 1)^p = \sum_{i=1}^{n} 2^p (c_i - c(\omega) - 1)^p.$$

Given that C was discarded, we know that $\sum_{i=1}^{n} (c_i - c_i(\omega) - 1) > \alpha^p$. Consequently,

$$\min index(\mathcal{C}) > (2\alpha)^p.$$

This leads to all $\mathcal{C}$ that were discarded by $\mathcal{HR}^3(\alpha)$ also being discarded by $\mathcal{HR}^3(2\alpha)$. Proving our lemma is consequently equivalent to showing that there is a hypercube $C' \in \mathcal{H}(\alpha, \omega)$ that is such that one of the 2^n cubes $\mathcal{Q}$ it is split into gets discarded by $\mathcal{HR}^3(2\alpha)$. An example of such a case for $p = 2$ and $n = 2$ is shown in Figure 5.4a and Figure 5.4b. For $\alpha = 4$, the cubes that are adjacent to the corner and do not lie on the diagonal of the square are not excluded. Yet, for $\alpha = 8$, 2 of the hypercubes in which they are split are discarded.

Let $C = (c_1, ..., c_n) \notin \mathcal{H}(\omega, \alpha)$ while $C' = (c_1 - 1, ..., c_n) \in \mathcal{H}(\omega, \alpha)$. In the following, we show that the hypercube $\mathcal{Q} = (2c_1 - 1, 2c_2 + 1, ..., 2c_n + 1)$, which is one of the hypercubes that C' gets split into by virtue of its coordinates,[6] will be discarded by $\mathcal{HR}^3(2\alpha)$, i.e., $\mathcal{Q} \notin \mathcal{H}(\omega, 2\alpha)$.

We know that $C = (c_1, ..., c_n)$ gets discarded, i.e., $\sum_{i=1}^{n} (c_i - c_i(\omega) - 1)^p > \alpha^p$.

Now, $\min index(\mathcal{Q}) = (2c_1 - (2c_1(\omega) + 1) - 1)^p + \sum_{i=2}^{n} (2c_i + 1 - (2c_i(\omega) + 1) - 1)^p$.

Consequently, $\min index(\mathcal{Q}) = 2^p (c_1 - c_1(\omega) - 1)^p + \sum_{i=2}^{n} [2(c_i - c_i(\omega) - 1) + 1]^p$.

6 Note that $2c_1 - 1 = 2(c_1 - 1) + 1$.

This value is obviously larger than $\sum_{i=1}^{n}[2(c_i - c_i(\omega) - 1)]^p$. From the premise that $\sum_{i=1}^{n}(c_i - c_i(\omega) - 1)^p > \alpha^p$, we can finally infer that $\min index(\mathcal{Q}) > (2\alpha)^p$. Thus, we can conclude that $\forall \alpha > 1\ V(\mathcal{H}(\alpha, \omega)) > V(\mathcal{H}(2\alpha, \omega))$. □

One of the consequences of Lemma 5.2 w.r.t. $RRR(\mathcal{HR}^3, \alpha)$, i.e., the RRR achieved by $\mathcal{HR}^3$ when the granularity is set to α, is

$$\forall \alpha > 1 : \ RRR(\mathcal{HR}^3, \alpha) > RRR(\mathcal{HR}^3, 2\alpha). \tag{5.8}$$

Note that this inequality is not sufficient to prove that we can achieve any RRR greater than 1, as series can converge to any real number. Consequently, we still need to show the following:

Lemma 5.4. $\lim_{\alpha \to \infty} RRR(\mathcal{HR}^3, \alpha) = 1$.

Proof. The cubes that are not discarded by $\mathcal{HR}^3(\alpha)$ are those for which $(|c_i - c_i(\omega)| - 1)^p \leqslant \alpha^p$. When $\alpha \to \infty$, Δ becomes infinitesimally small, leading to the cubes being single points. Each cube C thus contains a single point x with coordinates $x_i = c_i\Delta$. Especially, $c_i(\omega) = \omega$. Consequently,

$$\sum_{i=1}^{n}(|c_i - c_i(\omega)| - 1)^p \leqslant \alpha^p \leftrightarrow \sum_{i=1}^{n}\left(\frac{|x_i - \omega_i| - \Delta}{\Delta}\right)^p \leqslant \alpha^p. \tag{5.9}$$

Given that $\theta = \Delta\alpha$, we get

$$\sum_{i=1}^{n}\left(\frac{|x_i - \omega_i| - \Delta}{\Delta}\right)^p \leqslant \alpha^p \leftrightarrow \sum_{i=1}^{n}(|x_i - \omega_i| - \Delta)^p \leqslant \theta^p. \tag{5.10}$$

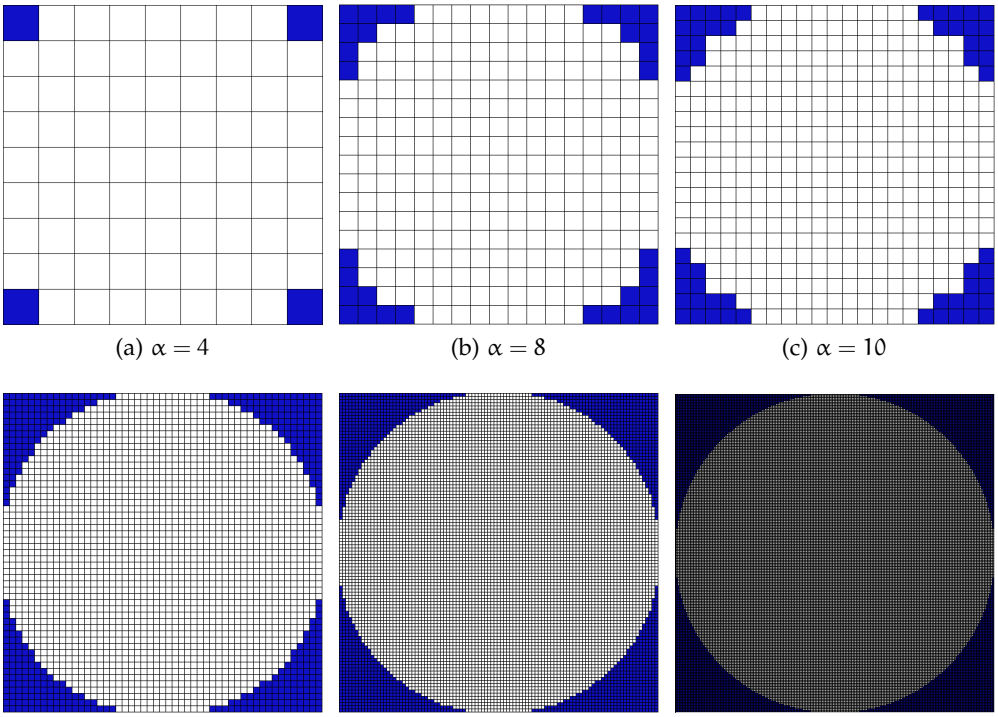
Finally, $\Delta \to 0$ when $\alpha \to \infty$ leads to

$$\sum_{i=1}^{n}(|x_i - \omega_i| - \Delta)^p \leqslant \theta^p \wedge \alpha \to \infty \to \sum_{i=1}^{n}|x_i - \omega_i|^p \leqslant \theta^p. \tag{5.11}$$

This is exactly the condition for linking specified in Definition 4 applied to Minkowski distances of order p. Consequently, $\mathcal{H}(\omega, \infty)$ is exactly $H(\omega, \theta)$ for any θ. Thus, the number of comparisons carried out by $\mathcal{HR}^3(\alpha)$ when $\alpha \to \infty$ is exactly C_{min}, which leads to the conclusion $\lim_{\alpha \to \infty} RRR(\mathcal{HR}^3, \alpha) = 1$. □

Our conclusion is illustrated by Figure 5.4, which shows the approximations computed by $\mathcal{HR}^3$ for different values of α with $p = 2$ and $n = 2$. The higher α, the closer the approximation is to a circle. Note that these results allow to conclude that for any RRR-value r larger than 1, there is a setting of $\mathcal{HR}^3$ that can compute links with a RRR smaller or equal to r.

An evaluation of the approach against space tiling as implemented by HYPPO revealed that using the setting $\alpha = 4$ is sufficient to achieve better runtimes as well a higher reduction ratio. Several parallel implementations of the approach are presented in (Ngonga Ngomo et al., 2013).



(a) $\alpha = 4$ (b) $\alpha = 8$ (c) $\alpha = 10$

(d) $\alpha = 25$ (e) $\alpha = 50$ (f) $\alpha = 100$

Figure 5.4: Approximation generated by $\mathcal{HR}^3$ for different values of α. The white squares are selected whilst the colored ones are discarded.

5.4 IMPLEMENTATION

The $\mathcal{HR}^3$ algorithm was implemented as shown in Algorithm 5.1. It is important to notice that the memory requirements of $\mathcal{HR}^3$ are smaller than those of most other approaches and especially than those of simple space tiling for any α such that $n(\alpha - 1)^p > \alpha^p$, as $\mathcal{HR}^3$ then generates less hypercubes. Yet, $\mathcal{HR}^3$ requires one supplementary computational step as it has to compute the index of cubes before discarding the unnecessary ones. Consequently, although we have shown that $\mathcal{HR}^3$ can achieve any RRR > 1, the question that remains to elucidate is whether this theoretical guarantee also offers a practically superior algorithm w.r.t. its runtime. That is the goal of the subsequent evaluation.

5.5 EVALUATION

5.5.1 *Experimental Setup*

We carried out four experiments to compare $\mathcal{HR}^3$ with LIMES 0.5's HYPPO and SILK 2.5.1. In the first and second experiments, we aimed to deduplicate DBpedia places by comparing their names (`rdfs:label`), minimum elevation, elevation and maximum elevation. We retrieved 2988 entities that possessed all four properties. We use the Euclidean metric on the last three values with the thresholds 49 meters resp. 99 meters for the first resp. second experiment. The third and fourth experiments aimed to discover links between Geonames and LinkedGeoData. Here, we compared the labels (`rdfs:label`), longitude and latitude of the instances. This

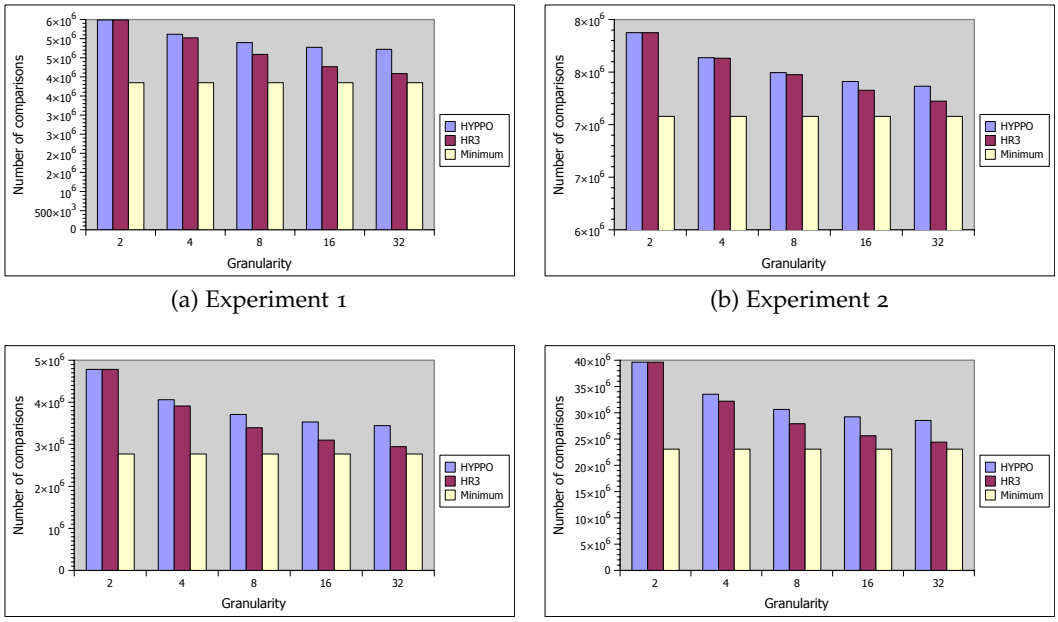
Algorithm 5.1 The $\mathcal{HR}^3$ algorithm

Require: Source knowledge base S, target knowledge base T, distance threshold θ, Minkowski distance δ of order p, granularity factor α
  Mapping $M := \emptyset$
  $\Delta = \theta/\alpha$
  for $\omega \in S \cup T$ **do**
    $C(\lfloor \omega_1/\Delta \rfloor, ..., \lfloor \omega_n/\Delta \rfloor) := C(\lfloor \omega_1/\Delta \rfloor, ..., \lfloor \omega_n/\Delta \rfloor) \cup \{\omega\}$
  end for
  for $s \in S$ **do**
    for $C \in \mathcal{H}(s, \alpha)$ **do**
      for $t \in C \cap T$ **do**
        if $\delta(s, t) \leqslant \theta$ **then**
          $M := M \cup (s, t, \delta(s, t))$
        end if
      end for
    end for
  end for
  return M

experiment was of considerably larger scale than the first one, as we compared 74458 entities in Geonames with 50031 entities from LinkedGeoData. Again, we measured the runtime necessary to compare the numeric values when comparing them by using the Euclidean metric. We set the distance thresholds to 1 resp. 9° in experiment 3 resp. 4. We ran all experiments on the same Windows 7 Enterprise 64-bit computer with a 2.8GHz i7 processor with 8GB RAM. The JVM was allocated 7GB RAM to ensure that the runtimes were not influenced by swapping. Only one of the kernels of the processors was used. Furthermore, we ran each of the experiments three times and report the best runtimes in the following.

5.5.2 *Results*

We first measured the number of comparisons required by HYPPO and $\mathcal{HR}^3$ to complete the tasks at hand (see Figure 5.5). Note that we could not carry out this section of the evaluation for SILK2.5.1 as it would have required altering the code of the framework. In the experiments 1, 3 and 4, $\mathcal{HR}^3$ can reduce the overhead in comparisons (i.e., the number of unnecessary comparisons divided by the number of necessary comparisons) from approximately 24% for HYPPO to approximately 6% (granularity = 32). In experiment 2, the overhead is reduced from 4.1% to 2%. This difference in overhead reduction is mainly due to the data clustering around certain values and the clusters having a radius between 49 meters and 99 meters. Thus, running the algorithms with a threshold of 99 meters led to only a small a-priori overhead and HYPPO performing remarkably well. Still, even on such data distributions, $\mathcal{HR}^3$ was able to discard even more data and to reduce the number of unnecessary computations by more than 50% relative. In the best case (Exp. 4, $\alpha = 32$, see Figure 5.5d), $\mathcal{HR}^3$ required approximately 4.13×10^6 less comparisons than HYPPO for $\alpha = 32$. Even for the smallest setting (Exp. 1, see Figure 5.5a), $\mathcal{HR}^3$ still required 0.64×10^6 less comparisons.



(a) Experiment 1

(b) Experiment 2

(c) Experiment 3

(d) Experiment 4

Figure 5.5: Number of comparisons for $\mathcal{HR}^3$ and HYPPO

We also measured the runtimes of SILK, Hyppo and $\mathcal{HR}^3$. The best runtimes of the three algorithms for each of the tasks is reported in Figure 5.6. Note that SILK's runtimes were measured without the indexing time, as the data fetching and indexing are merged to one process in SILK. Also note that in the second experiment, SILK did not terminate due to higher memory requirements. We approximated SILK's runtime by extrapolating approximately 11 min it required for 8.6% of the computation before the RAM was filled. Again, we did not consider the indexing time.

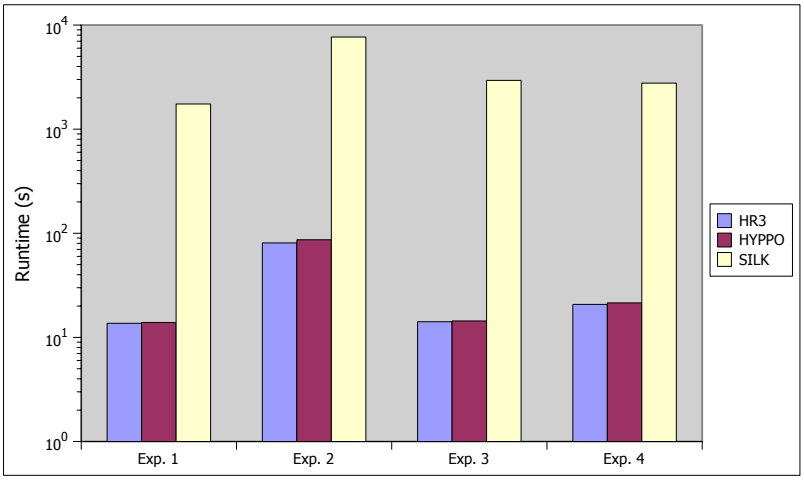
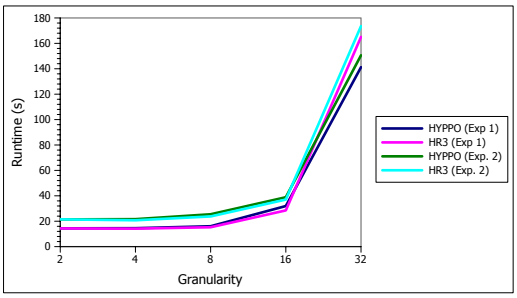


Figure 5.6: Comparison of the runtimes of $\mathcal{HR}^3$, Hyppo and SILK2.5.1

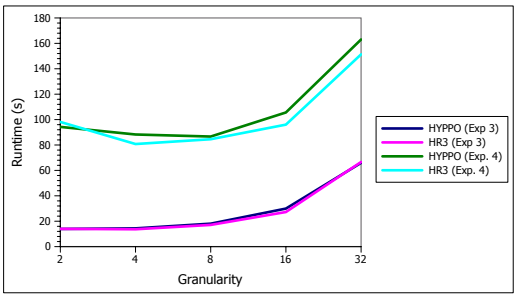
Due to the considerable difference in runtime (approximately 2 orders of magnitude) between Hyppo and SILK, we report solely Hyppo and $\mathcal{HR}^3$'s runtimes in the detailed runtimes figures, i.e., Figure 5.7a and Figure 5.7b. Overall, $\mathcal{HR}^3$ out-

performed the other two approaches in all experiments, especially for $\alpha = 4$. It is important to note that the improvement in runtime increases with the complexity of the experiment. For example, while $\mathcal{HR}^3$ outperforms HYPPO by 3% in the second experiment, the different grows to more than 7% in the fourth experiment. In addition, the improvement in runtime augments with the threshold. This can be seen in the third and fourth experiments. While $\mathcal{HR}^3$ is less than 2% faster in the third experiment, it is more than 7% faster when $\theta = 4$ the fourth experiment . As expected, $\mathcal{HR}^3$ is slower than HYPPO for $\alpha < 4$ as it carries out exactly the same comparisons but still has the overhead of computing the index. Yet, given that we know that $\mathcal{HR}^3$ is only better when $n(\alpha-1)^p > \alpha^p$, our implementation only carries out the indexing when this inequality holds. By these means, we can ensure that $\mathcal{HR}^3$ is only used when it is able to discard hypercubes that HYPPO would not discard, therewith reaching superior runtimes both with small and large values α. Note that the difference between the improvement of the number of comparisons necessitated by $\mathcal{HR}^3$ and the improvement in runtime over all experiments is due to the supplementary indexing step required by $\mathcal{HR}^3$.

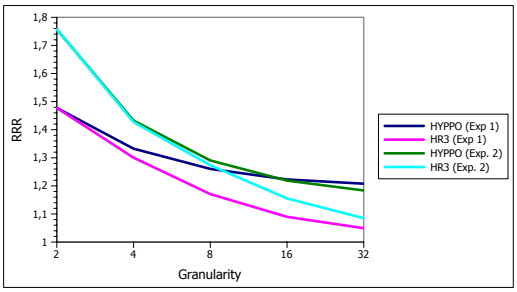
Finally, we measured the RRR of both $\mathcal{HR}^3$ and HYPPO (see Figure 5.7c and Figure 5.7d). In the two-dimensional experiments 3 and 4, HYPPO achieves a RRR close to 1. Yet, it is still outperformed by $\mathcal{HR}^3$ as expected. A larger difference between the RRR of $\mathcal{HR}^3$ and HYPPO can be seen in the three-dimensional experiments, where the RRR of both algorithms diverge significantly. Note that the RRR difference grows not only with the number of dimensions but also with the size of the problem. The difference in RRR between HYPPO and $\mathcal{HR}^3$ does not always reflect the difference in runtime due to the indexing overhead of $\mathcal{HR}^3$. Still, for $\alpha = 4$, $\mathcal{HR}^3$ generates a sufficient balance of indexing runtime and comparison runtime (i.e., RRR) to outperform HYPPO in all experiments.



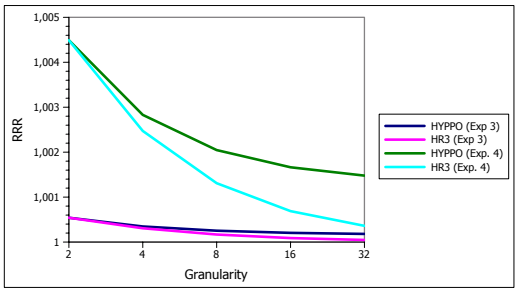
(a) Runtimes for experiments 1 and 2



(b) Runtimes for experiments 3 and 4



(c) RRR for experiments 1 and 2



(d) RRR for experiments 3 and 4

Figure 5.7: Comparison of runtimes and RRR of $\mathcal{HR}^3$ and HYPPO

5.6 RELATED WORK

The growing size and number of knowledge bases available in the Linked Data Cloud makes Link Discovery intrinsically complex with respect to its runtime. To address this issue, manifold time-efficient frameworks have been developed. LIMES (Ngonga Ngomo, 2011) offers a complex grammar for link specifications that can be translated into a combination of time-efficient atomic mappers that are combined via a hybrid approach. For example, LIMES implements a dedicated approach for numeric values called HYPPO. SILK (Isele et al., 2011) implements a different Link Discovery paradigm and aims to place all instances that are to be compared in a multi-dimensional space. It then uses MultiBlock to discard unnecessary comparisons efficiently. In contrast to LIMES and SILK, which implement lossless approaches, the approach presented in (Song and Heflin, 2011) uses a candidate selection approach based on discriminative properties to compute links very efficiently but potentially loses links while doing so. Other frameworks and approaches include those described in (Glaser et al., 2009; Raimond et al., 2008; Scharffe et al., 2009).

Albeit Link Discovery is closely related with record linkage (Elmagarmid et al., 2007) and deduplication (Bleiholder and Naumann, 2008), it is important to notice that Link Discovery goes beyond these two tasks as Link Discovery aims to provide the means to link entities via arbitrary relations. Different blocking techniques such as standard blocking, sorted-neighborhood, bi-gram indexing, canopy clustering and adaptive blocking have been developed by the database community to address the problem of the quadratic time complexity of brute force comparison (Köpcke et al., 2009). In addition, very time-efficient approaches have been proposed to compute string similarities for record linkage, including AllPairs (Bayardo et al., 2007), PPJoin and PPJoin+ (Xiao et al., 2008). However, these approaches alone cannot deal with the diversity of property values found on the Web of Data as they cannot deal with numeric values. In addition, most time-efficient string matching algorithms can only deal with simple link specifications, which are mostly insufficient when computing links between large knowledge bases.

In recent work, the discovery of adequate link specifications has been addressed mainly by using machine learning approaches. For example, (Song and Heflin, 2011) detect discriminative properties by using string concatenations. RAVEN (Ngonga Ngomo et al., 2011) combines stable marriage algorithms and a perceptron-based learning algorithm with the frame of active learning to compute boolean and linear classifiers. SILK (Isele and Bizer, 2011, 2012) employs genetic programming to learn link configurations from positive and negative examples. (Ngonga Ngomo and Lyko, 2012) go a step further and combine genetic programming with active learning to discover high-accuracy link specificity with a small number of annotations. Another approach based on genetic programming is presented in (Nikolov et al., 2012). Here, the authors show how link specifications can be learned without any input from the user. To the best of our knowledge, none of the approaches presented previously provide formal guarantees w.r.t. their performance. $\mathcal{HR}^3$ is the first matching approach that it guaranteed not to lose links while converging to the small possible reduction ratio. Note that while $\mathcal{HR}^3$ was designed for numeric values, it can be used in any space with Minkowski distances, for example for comparing indexes in multi-dimensional spaces. Thus, it can be used for any datatype mapped to a metric space.

5.7 CONCLUSION AND FUTURE WORK

In this chapter, we presented $\mathcal{HR}^3$, a time-efficient approach for the discovery of links in spaces with Minkowski distances. We proved that our approach can achieve is optimal w.r.t its reduction ration by showing that its RRR converges towards 1 when α converges towards ∞. It is important to note that an optimal $\mathrm{RRR}(\mathcal{A})$ does not necessarily mean that $\mathcal{A}$ outperforms algorithms with a poorer RRR with respect to runtime as achieving a good RRR score usually requires better preprocessing (usually in form of indexing), which might be more time-demanding than the combination of a rougher preprocessing and a run with a poorer RRR. Thus, in addition to proving formally that we can guarantee a RRR that converges towards 1, we implemented our approach and compared it with the state-of-the-art algorithms HYPPO implemented in LIMES and SILK. We showed that we outperform both frameworks w.r.t. to their runtime and that we reach RRR close to 1 for α as small as 32. Our experiments also showed that $\alpha = 4$ is a good setup for $\mathcal{HR}^3$. Our approach aims to be the first of a novel type of Link Discovery approaches, i.e., of approaches which can guarantee theoretical optimality while also being empirically usable. In future work, we will thus aim to develop more of such approaches and to make use of their theoretical characteristics for memory and space management. With respect to $\mathcal{HR}^3$, we will mainly improve the implementation of its indexing to ensure even better runtimes.

6

REDUCTION-RATIO-OPTIMAL COMPUTATION OF GEO-SPATIAL DISTANCES

PREAMBLE

This chapter is based on (Ngonga Ngomo, 2013) and builds upon $\mathcal{HR}^3$. Like in the previous chapter, the optimization of the reduction ratio of LD algorithms is considered. However, orthodromic spaces are now the reference spaces within which LD is to be carried out. This work was carried out by the author alone.

6.1 INTRODUCTION

The Linked Open Data Cloud (LOD Cloud) has developed to a compendium of approximately 300 datasets over the last few years. Currently, geographic datasets contain approximately 6 billion triples and make up 19.4% of the triples in the LOD Cloud. Projects such as LinkedGeoData[1] promise an increase of these numbers by orders of magnitude in the near future. However, only 7.1% of the links between knowledge bases in the LOD Cloud currently connect geographic entities. This means that less than 1% of triples within the geographic datasets of the LOD Cloud are links between knowledge bases.[2] This blatant lack of links is partly due to two factors: First, it is due to the *large number of geo-spatial entities* available on the Linked Open Data Cloud. Moreover, the *geo-spatial resources* are often described as (often ordered) sets of points which describe geometric objects such as (multi-) polygons or (multi-) polylines. This way of describing resources differs considerably from the approach followed for most Linked Data resources, which are commonly easiest identified by the means of a label. Consequently, such descriptions have not yet been paid much attention to in the field of link discovery (LD).

We address this gap by presenting ORCHID, a reduction-ratio-optimal approach for LD. ORCHID assumes the LD problem as being formulated in the following way: Given a set S of source instances, a set T of target instances and a distance threshold θ, find the set of triples $(s, t, \delta(s, t)) \in S \times T \times \mathbb{R}^+$ such that $\delta(s, t) \leqslant \theta$. Given this assumption, the idea behind ORCHID is to address the LD problem on geographic data described as (ordered) sets of points by two means. First, ORCHID implements time-efficient algorithms for computing whether the distance between two polygons s and t is less or equal to a given distance threshold θ. Moreover, ORCHID implements a space tiling algorithm for orthodromic spaces which allows discarding yet another large number of unnecessary computations.

The rest of this chapter is structured as follows: In Section 6.2, we present the core notation used throughout this chapter as well as some formal considerations underlying our approach. Section 6.3 presents two approaches that allow comput-

1 See http://linkedgeodata.org. Last access: January 11th, 2013.

2 See http://wifo5-03.informatik.uni-mannheim.de/lodcloud/state/ for an overview of the current state of the Cloud. Last access: January 11th, 2013.

ing the Hausdorff distance between two polygons efficiently.[3] Subsequently, we present the space discretization approach implemented by ORCHID and show that it is optimal with respect to its reduction ratio. We then present a thorough evaluation of our approach on three datasets of different sizes and complexity. We also compare our approach with a state-of-the-art LD framework which implements the orthodromic distance. We conclude the chapter with a brief overview of related work (Section 6.6) and a discussion of our results (Section 6.7). The approach presented here was integrated in the LIMES framework.[4] Due to space restrictions, we had to omit some details of the approaches presented herein. These can be found in the corresponding technical report on the project webpage.

6.2 PRELIMINARIES

The formal specification of LD adopted herein is tantamount to the definition proposed in (Ngonga Ngomo, 2012b): Given a set S of source resources, a set T of target resources and a relation R, our goal is to find the set $M \subseteq S \times T$ of pairs $(s,t) \in S \times T$ such that $R(s,t)$. If R is `owl:sameAs`, then we are faced with a *deduplication task*. Given that the explicit computation of M is usually a very complex endeavor, M is usually approximated by a set $\tilde{M} = \{(s,t,\delta(s,t)) \in S \times T \times \mathbb{R}^+ : \delta(s,t) \leqslant \theta\}$, where δ is a distance function and $\theta \geqslant 0$ is a distance threshold. For geographic data, the resources s and t are described by using single points or (ordered) sets of points, which we regard as polygons. Given that we can regard points as polygons with one node, we will speak of resources being described as polygons throughout this chapter. We will use a subscript notation to label the nodes that make up resources. For example, if s had three nodes, we would denote them s_1, s_2, and s_3. For convenience's sake, we will write $s = \{s_1, s_2, s_3\}$ and $s_i \in s$.

While there are several approaches for computing the distance between two polygons (Atallah et al., 1991), a common approach is the use of the Hausdorff distance (Nutanong et al., 2011) hd:

$$hd(s,t) = \max_{s_i \in s}\{\min_{t_j \in t}\{\delta(s_i,t_j)\}\}, \tag{6.1}$$

where δ is the metric associated to the affine space within which the polygons are defined. We assume that the earth is a perfect ball with radius $R = 6378\,km$. Then, δ is the *orthodromic distance* and will be denoted od in the rest of this chapter. Given these premises, the LD task we investigate in this chapter is the following: Find the set $\tilde{M} = \{(s,t,hd(s,t)) \in S \times T : hd(s,t) \leqslant \theta\}$ where $\forall s_i \in s\ \forall t_j \in t\ \delta(s_i,t_j) = od(s_i,t_j)$. It is important to notice that the Hausdorff distance is a metric in the mathematical sense of the term in any metric space. Moreover, the orthodromic distance is also known to be a metric, leading to the problem formulated above being expressed in a metric space.

Two requirements are central for the approaches developed herein. First, the approaches have to be *complete* (also called lossless (Ngonga Ngomo, 2012b)), which simply means that they must be able to compute *all triples* $(s,t,hd(s,t)) \in S \times T \times \mathbb{R}^+$ for which $hd(s,t) \leqslant \theta$ holds. This characteristic is not fulfilled by certain blocking approaches, which trade runtime efficiency for completeness. In addition

3 The Hausdorff distance can be used to compare the distance between any two sets of ordered points located in a space where a distance function is defined. Thus, while we focus on polygons in this chapter, our approach can be used for all sets of points.

4 http://limes.sf.net

to developing a complete approach, we aim to develop a reduction-ratio-optimal approach (Ngonga Ngomo, 2012b): Let $\mathcal{A}$ be an algorithm for computing $\tilde{M}$ and α be the vector that contains all parameters necessary to run $\mathcal{A}$. Moreover, let $|\mathcal{A}(\alpha)|$ be the number of computations of hd carried out by $\mathcal{A}$ when assigned the vector of parameters α. We call $\mathcal{A}(\alpha)$ reduction-ratio-optimal when

$$\forall r < 1 - \frac{|\tilde{M}|}{|S||T|} \; \exists \alpha : 1 - \frac{|\mathcal{A}(\alpha)|}{|S||T|} \geqslant r. \tag{6.2}$$

Naive approaches to computing $\tilde{M}$ have two drawbacks: First, they require $|s||t|$ calls of od to compute $hd(s,t)$. Moreover, they carry out $|S||T|$ computations of hd to find all elements of $\tilde{M}$. Addressing the time complexity of LD on geographic data thus requires addressing these two quadratically complex problems. Our approach addresses the time complexity of the first problem by making use of the Cauchy-Schwarz inequality, i.e.,

$$od(x,y) \leqslant od(x,z) + od(z,y), \tag{6.3}$$

and of bounding circles for approximating the distance between polygons. The second problem is addressed by the means of a reduction-ratio-optimal tiling approach similar to the $\mathcal{HR}^3$ algorithm (Ngonga Ngomo, 2012a).

6.3 EFFICIENT COMPUTATION OF HAUSDORFF DISTANCES

Several approaches have addressed the time-efficient computation of Hausdorff distances throughout literature (see (Nutanong et al., 2011) for a good overview). Yet, so far, these approaches have not been concerned with the problem of only finding those triples $(s,t,hd(s,t))$ with $hd(s,t) \leqslant \theta$. In the following, we present several approaches for achieving this goal. These approaches are later evaluated in Section 6.5. For space reasons, we omit the pseudo-code for the first two approaches. These can be found in the technical report.

6.3.1 *Naive approach*

The naive approach for computing $hd(s,t)$ would compare all elements of the polygon $s \in S$ with all elements of the polygons $t \in T$ by computing the orthodromic distance between all $s_i \in s$ and $t_j \in t$. Let $\bar{S}$ be the average size of the polygons in S and $\bar{T}$ be the average size of the polygons in T. The best- and worst-case runtime complexities of the naive approach are then $O(|S||T|\bar{S}\bar{T})$.

6.3.2 *Bound approach*

A first idea to make use of the bound $hd(s,t) \leqslant \theta$ on distances lies in the observation that

$$\exists s_i \in S : \min_{t_j \in t}\{od(s_i,t_j)\} > \theta \rightarrow hd(s,t) > \theta \tag{6.4}$$

This insight allows terminating computations that would not lead to pairs for which $hd(s,t) \leqslant \theta$ by terminating the computation as soon as a s_i is found that fulfils Equation 6.4. In the best case, only one point of each $s \in S$ is compared to all points of $t \in T$ before the computation of $hd(s,t)$ is terminated. Thus, the best-case

Algorithm 6.1 Naive implementation of bound Hausdorff distances

```
1: max ← 0
2: for s_i ∈ s do
3:    min ← ∞
4:    for t_j ∈ t do
5:       d = od(s_i, t_j)
6:       if d < min then
7:          min ← d
8:       end if
9:    end for
10:   if max < min then
11:      max ← min
12:   end if
13: end for
14: if max ⩽ θ then
15:   return max
16: else
17:   return ∅
18: end if
```

complexity of the approach is $O(|S||T|\bar{T})$. In the worst case (i.e., in the case that the set of mappings returned is exactly $S \times T$), the complexity of the bound approach is the same as that of the naive approach, i.e., $O(|S||T|\bar{S}\bar{T})$.

6.3.3 *Indexed Approach*

The indexed approach combines the intuition behind the bound approach with geometrical characteristics of the Hausdorff distance by using two intuitions. The first intuition is that if the minimal distance between any point of s and any point of t is larger than θ, then $hd(s,t) > \theta$ must hold. Our second intuition makes use of the triangle inequality to approximate the distances $od(s_i, t_k)$. In the following, we present these two intuitions formally. We dub the indexed approach which relies on the second intuition alone CS while we call the indexed approach that relies on both intuitions $BC + CS$.

6.3.3.1 *Intuition 1: Bounding Circles*

Formally, the first intuition can be expressed as follows:

$$\min_{s_i \in s,\ t_j \in t} \{od(s_i, t_j)\} > \theta \rightarrow hd(s,t) > \theta. \tag{6.5}$$

Finding the two points s_i and t_j which minimize the value of $od(s_i, t_j)$ requires $O(|s||t|)$ computations of od, i.e., $O(|S||T|\bar{S}\bar{T})$ overall. However, a lower bound for this minimum for all pairs $(s,t) \in S \times T$ can be computed efficiently by using encompassing circles: Let $\mathcal{C}(s)$ resp. $\mathcal{C}(t)$ be the smallest circles that fully encompass s resp. t. Moreover, let $r(s)$ resp. $r(t)$ be the radius of these circles and $\zeta(s)$ resp. $\zeta(t)$ be the centers of the circles $C(s)$ resp. $C(t)$. Then,

$$\min_{s_i \in s, t_j \in t} \{od(s_i, t_j)\} > od(\zeta(s), \zeta(t)) - (r(s) + r(t)) = \mu(s,t). \tag{6.6}$$

Algorithm 6.2 Bound implementation of bound Hausdorff distances

```
1: max ← 0
2: for s_i ∈ s do
3:    min ← ∞
4:    for t_j ∈ t do
5:       d = od(s_i, t_j)
6:       if d < min then
7:          min ← d
8:       end if
9:    end for
10:   if min > θ then
11:      return ∅
12:   end if
13:   if max < min then
14:      max ← min
15:   end if
16: end for
17: if max ⩽ θ then
18:    max
19: else
20:    return ∅
21: end if
```

Figure 6.1 displays the intuition behind this approximation graphically. Note that this equation also holds when the circles overlap (in which case $od(\zeta(s), \zeta(t)) - (r(s) + r(t)) < 0$ as $od(\zeta(s), \zeta(t)) < (r(s) + r(t))$.

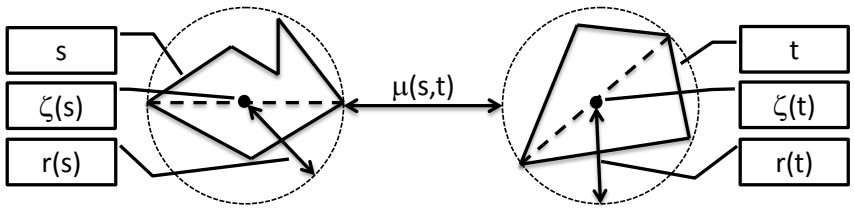


Figure 6.1: Lower bound of Hausdorff distances based on circles

Computing the smallest circle that encompasses any polygon x can be carried out in $O(|x|^2)$ by simply computing $od(x_i, x_k)$ for all $(x_i, x_k) \in x^2$. Then,

$$r(x) = \frac{\max\limits_{x_i \in x, x_k \in x} od(x_i, x_k)}{2} \tag{6.7}$$

while

$$\zeta(x) = \frac{x^+ + x^-}{2} \text{ where } (x^+, x^-) = \underset{x_i \in x, x_k \in x}{\arg\max}\ od(x_i, x_k). \tag{6.8}$$

The proof that the radius $r(x)$ must have the value shown in Equation 6.7 is as follows: The points within a circle with radius r' are at most at a distance $2r'$ of each other. Consequently, any circle with radius $r' < r(x)$ cannot contain both elements of the pair $(x^+, x^-) = \underset{x_i \in x, x_k \in x}{\arg\max}\ od(x_i, x_k)$. Thus, the smallest possible

radius of a circle that encompasses x fully must be the maximal distance between points which belong to x. This is exactly the value of $r(x)$. Now the only way to ensure that a circle with radius $r(x)$ really encompasses all points in x is to have x^+ and x^- to be diametrically opposite. Thus, $\zeta(x)$ must be exactly in the middle of x^+ and x^-.

The runtime complexity of this approximation is $O(|S|\bar{S}^2 + |T|\bar{T}^2 + |S||T|)$. $O(|S|\bar{S}^2 + |T|\bar{T}^2)$ computations of od are required to determine the circles and their radii while $O(|S||T|)$ computations are required to compare the circles computed out of S with those from T. Note that for large problem $\bar{S}^2$ resp. $\bar{T}^2$ are very small compared to $|S|$ resp. $|T|$, leading to $O(|S|\bar{S}^2 + |T|\bar{T}^2 + |S||T|) \approx O(|S| + |T| + |S||T|) \approx O(|S||T|)$.

6.3.3.2 *Intuition 2: Distance Approximation using the Cauchy-Schwarz Inequality*

Now given that we have computed all distances between all pairs $(t_j, t_k) \in t^2$, we can reuse this information to approximate distances from any s_i to any t_k by relying on the Cauchy-Schwarz inequality in a fashion similar to the LIMES algorithm presented in (Ngonga Ngomo and Auer, 2011). The idea here is that we can compute an upper and a lower bound for the distance $od(s_i, t_k)$ by using the distance $od(s_i, t_j)$ previously computed as follows:

$$|od(s_i, t_j) - od(t_j, t_k)| \leqslant od(s_i, t_k) \leqslant od(s_i, t_j) + od(t_j, t_k). \tag{6.9}$$

For each s_i, exploiting these pre-computed distances can be carried out as follows: For all t_k for which $od(s_i, t_k)$ is unknown, we approximate the distance from s_i to t_k by finding a point t_j for which

$$t_j = \arg\min_{t_x \in t'} od(t_x, t_k) \tag{6.10}$$

holds, where $t' \subseteq t$ is the set of points t_x of t for which $od(s_i, t_x)$ is known. We call the point t_j an *exemplar* for t_k. The idea behind using one of points closest to t_k is that it gives us the best possible lower bound $|od(s_i, t_j) - od(t_j, t_k)|$ for the distance $od(s_i, t_k)$. Now if $|od(s_i, t_j) - od(t_j, t_k)| > \theta$, then we can discard the computation of the distance $od(s_i, t_k)$ and simply assign it any value $\Theta > \theta$. Moreover, if $|od(s_i, t_j) - od(t_j, t_k)|$ is larger than the current known minimal distance between s_i and points in t, then we can also discard the computation of $od(s_i, t_k)$. If such an exemplar does not exist or if our approximations fail to discard the computation, then only do we compute the real value of the distance $od(s_i, t_k)$.

The best-case complexity of this step alone would be $O(|S||T|\bar{S})$ while in the worst case, we would need to carry out $O(|S||T|\bar{S}\bar{T})$ computations of od. The overall complexity of the indexed approach is $O(|S|\bar{S}^2 + |T|\bar{T}^2 + |S||T|)$ (i.e., that of the bounding circles filter) in the best case and $O(|S|\bar{S}^2 + |T|\bar{T}^2 + |S||T| + |S||T|\bar{S}\bar{T})$ in the worst case. The overall algorithm underlying the indexed approach is shown in Algorithm 6.3.

6.4 ORCHID

Although the indexed method presented above can significantly reduce the number of computations carried out to compare S and T, it still needs at least $|S||T|$ comparisons. For example, imagine our source and target datasets were all geo-spatial entities on the portion of the surface of the planet shown in Figure 6.2. If

Algorithm 6.3 Implementation of the BC + CS Hausdorff distance computation. The implementation of CS lacks lines 1,2,3 and 28.

```
1: if (od(c(s), c(t)) − r(s) − r(t) > θ) then
2:   return ∅
3: else
4:   max ← 0
5:   for s_i ∈ s do
6:     min ← ∞
7:     for t_j ∈ t do
8:       e = exemplar(t_j)
9:       if e ≠ ∅ then
10:        approx = |od(s_i, e) − od(e, t_j)|
11:        if approx > θ ∨ approx > min then
12:          d(s_i, t_j) = θ + 1
13:        else
14:          d(s_i, t_j) = od(s_i, t_j)
15:        end if
16:      else
17:        d(s_i, t_j) = od(s_i, t_j)
18:      end if
19:      min = min(min, d(s_i, t_j))
20:    end for
21:    max = max(max, min)
22:  end for
23:  if max > θ then
24:    return ∅
25:  else
26:    return max
27:  end if
28: end if
```

Oslo (which has the coordinates (59°56′58″ N, 10°45′23″ E)) was the resource to link via `dbp:near`, then the approaches above would compare it with each of the other elements of the dataset. The idea behind ORCHID is to reduce the number of comparisons even further while remaining complete and being reduction-ratio-optimal. To achieve this goal, ORCHID uses a space discretization approach and only compares polygons $t \in T$ which lie within a certain range of $s \in S$. An example of the discretization generated by ORCHID is shown in Figure 6.2. Instead of comparing Oslo with all other elements of the dataset, ORCHID would only compare it with the geo-spatial objects shown in the gray cells. In the following, we present ORCHID formally and prove that it is both complete and reduction-ratio-optimal.

6.4.1 *Preliminaries*

Explaining the approach implemented by ORCHID prerequisites the explication of a set of characteristics of the orthodromic distance od. Given a polygon s, finding all points t such that $hd(s,t) \leqslant \theta$ requires being able to find all points y for which

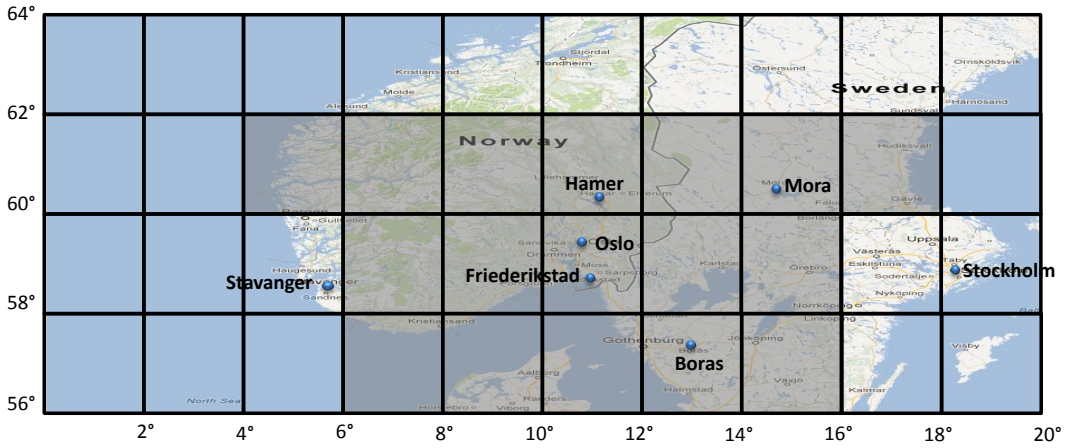


Figure 6.2: Example of tiling for $\alpha = 1$ and $\theta = 222.6\,\mathrm{km}$ (i.e., $\Delta_R = 2°$). Here, the resource to link is Oslo. The gray cells are the elements of A(Oslo).

$od(x, y) \leqslant \theta$ given a point x. In general, a point x on the surface of the planet can be characterized by two values: its latitude $lat(x)$ and its longitude $lon(x)$. These two values are bound as $-\pi/2 \leqslant lat(x) \leqslant \pi/2$ and $-\pi \leqslant lon(x) \leqslant \pi$ always hold.[5] We write $x = (lat(x), lon(x))$ to denote points. Now, given a point y with $lon(x) = lon(y)$, then $od(x, y) = R|lat(x) - lat(y)|$. Yet, if $lat(x) = lat(y)$, then $od(x, y) = R|lon(x) - lon(y)|cos(lat(x))$. This difference between latitude and longitude is central when finding all points y for which $od(x, y) \leqslant \theta$. Formally, it means that we can create a discretization in which we treat the latitude values independently from the longitude values but not the other way around. This particular characteristic of latitude and longitude values lies at the heart of ORCHID.

6.4.2 *Discretization for Geo-Spatial Points*

The idea behind ORCHID is to make use of the values of latitude and longitude being bound to first create a grid on the surface of the planet. We call $\alpha \in \mathbb{N}$ the granularity parameter of ORCHID. Given the premises described in Section 6.4.1, we can infer that for $x \in s \in S$ and $t \in T \in \mathsf{T}$

$$od(x, y) \leqslant \theta \rightarrow |lat(x) - lat(y)| \leqslant \theta/R = \theta_R. \tag{6.11}$$

Based on this equation, we can create a grid such that width and height of each cell of the grid is $\Delta_R = \theta_R/\alpha$. For each cell c_i, two whole numbers c_i^{lat} and c_i^{lon} exist such that c_i contains only points x for which

$$(c_i^{lat}\Delta \leqslant lat(x) < (c_i^{lat} + 1)\Delta) \wedge (c_i^{lon}\Delta \leqslant lon(x) < (c_i^{lon} + 1)\Delta) \tag{6.12}$$

holds. We call (c_i^{lat}, c_i^{lon}) the coordinates of c_i. Moreover, we write $x \in c_i$ if Equation 6.12 holds for x. We also write $c_i(x)$ to signify the cell to which x belongs. In our example (Figure 6.2), the cell which contains Oslo has the coordinates $(29, 5)$. Given this definition of a grid, the set $\tilde{M}(x)$ of y with $od(x, y) \leqslant \theta$ is clearly a subset of all y for which $|lat(x) - lat(y)| \leqslant \theta_R$ holds. With respect to our grid, we can infer the following inequality:

$$x \in c_i \wedge y \in c_j \wedge |c_i^{lat} - c_j^{lat}| > \alpha \rightarrow y \notin \tilde{M}(x). \tag{6.13}$$

5 All angles in this chapter are assumed to be in radian unless stated otherwise.

We call the set of all cells which abide by this inequality $LAT(x)$. Finding a similar equation for longitudes is more demanding, as the equation depends on the latitude of cells c_i and c_j. Formally, the set $\tilde{M}(x)$ of y with $od(x,y) \leqslant \theta$ is clearly a subset of all y for which $|lat(x) - lat(y)| \leqslant \theta_R / \min\{\cos(lon(x), lon(y))\}$ holds. Consequently, we can derive the following equation:

$$x \in c_i \wedge y \in c_j \wedge |c_i^{lon} - c_j^{lon}| > \left\lceil \frac{\alpha}{mincos(c_i, c_j)} \right\rceil \rightarrow y \notin \tilde{M}(x) \tag{6.14}$$

where

$$mincos(c_i, c_j) = \min\{\cos(\alpha c_i), \cos(\alpha(c_i + 1)), \cos(\alpha(c_j)), \cos(\alpha(c_j + 1)). \tag{6.15}$$

We call this set $LON(x)$. Now, if one the minimal cosine values in Equation 6.15 is 0, then Equation 6.14 is not well-defined. This happens when one of the cells c_i or c_j is adjacent to one of the poles. In this case, we assume $\frac{\alpha}{mincos(c_i,c_j)} = 0$. This assumption has the simple consequence that we select all cells c_j at the poles to contain potential y with $od(x,y) \leqslant \theta$. We can now generate a first approximation of $\tilde{M}(x)$ by computing the intersection of all y that abide by Equation 6.13 and Equation 6.14. We call this set $A(x) = LAT(x) \cap LON(x)$. Note that $\tilde{M}(x) \subseteq A(x)$. An example of such a set is shown in Figure 6.2 where A(Oslo) is depicted as a set of grey squares. Note that given that $\alpha = 1$, we only need to consider the cells with $28 \leqslant c_i^{lat} \leqslant 30$. Yet, given that $\cos(60°) = 0.5$, the number of cells that have to be considered in longitude grows from 5 to 7 when crossing the 60th north parallel.

6.4.3 *Optimality of Orchid for points*

While it is guaranteed that $\tilde{M}(x) \subseteq A(x)$, it is possible that $A(x)$ contains grid cell c with $\forall y \in c, (x, y, od(x,y)) \notin \tilde{M}$.[6] Such cells must be eliminated from $A(x)$ as they lead to unnecessary comparisons. Achieving this goal can be carried out by measuring the minimal distance from the cell $c(x)$ which contains x and all other cells $c \in A(x)$. Let us assume that c is at the north east of $c(x)$ (for reasons of symmetry, the argumentation can be extended to all other cells). In our example, such a cell would be that which contains Mora. Then the most north eastern point of $ne(c(x))$ has the coordinates $\Delta(c_i^{lat}(x) + 1, c_i^{lon}(x) + 1)$ while the most south western point of $sw(c)$ of c has the coordinates $\Delta(c_i^{lat}, c_i^{lon})$. Consequently, the minimal distance from points in $c(x)$ to points in c is

$$\min_{od}(c(x), c) = od(\Delta(c_i^{lat}(x) + 1, c_i^{lon}(x) + 1), \Delta(c_i^{lat}(x), c_i^{lon})). \tag{6.16}$$

We thus define the set $OPT(x) \subseteq A(x)$ as

$$OPT(x) = \{y \in A(x) : \min_{od}(c(x), c(y)) \leqslant \theta\}. \tag{6.17}$$

This set is guaranteed not to contain any cell with which elements of $c(x)$ should be compared. Consequently, it is the set of points x and all other elements of $c(x)$ are compared to by ORCHID.

$OPT(x)$ is optimal in the sense that

$$\lim_{\alpha \rightarrow +\infty} OPT(x) = \tilde{M}(x). \tag{6.18}$$

6 Note that $od(x,y) = hd(x,y)$ for $|x| = |y| = 1$.

This is simply due to $\alpha \to +\infty$ leading to $\Delta \to 0$. In this case, $c(x) = \{x\}$ and $c(y) = \{y\}$. Thus, $\min_{od}(c(x), c(y)) = od(x, y)$ which allows to infer that $OPT(x) = \tilde{M}(x)$ from Equation 6.17. Note that this proof shows that ORCHID fulfils a necessary and sufficient condition to be reduction-ratio-optimal on single points in the sense of (Ngonga Ngomo, 2012a). In our example $A(Oslo) = OPT(Oslo)$.

6.4.4 *Comparing Polygons with Orchid*

The extension of $OPT(x)$ to polygons is based on the following observation: Given the definition of Hausdorff distances,

$$hd(s, t) \leqslant \theta \to \forall s_i \in s \; \exists t_j \in t : od(s_i, t_j) \leqslant \theta \tag{6.19}$$

holds. Consequently, $OPT(s) = \bigcap_{s_i \in s} OPT(s_i)$. The reduction-ratio optimality of ORCHID for polygons follows from its reduction-ratio-optimality for points.

6.5 EVALUATION

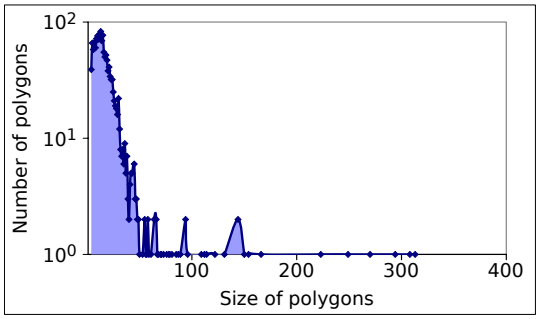
The goal of the evaluation was to assess the performance of our approaches with respect to their runtime and the number of computation of the orthodromic distance that they carried out. To achieve this goal, we first compare the naive, bound, CS and $BC + CS$ implementations of the computations of bound Hausdorff distance on samples from three different datasets. Note that we refrained from using the whole datasets because the runtime of the naive approach would have been impracticable. In the second part of our evaluation, we study the combination of ORCHID and our Hausdorff implementations.

6.5.1 *Experimental Setup*

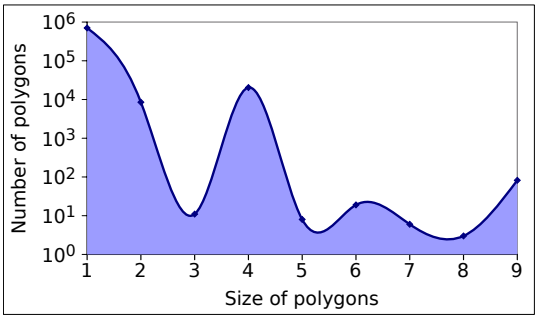
We selected three publicly available datasets of different sizes for our experiments. The first dataset, *Nuts*, contains a detailed description of 1,461 specific European regions.[7] The second dataset, *DBpedia*, contains all 731,922 entries from DBpedia that possess a geometry entry.[8] Finally, the third dataset, LGD, contains all 3,836,119 geo-spatial objects from LinkedGeoData that are instances of the class `Way`.[9] An overview of the distribution of the polygon sizes in these datasets is given in Figure 6.3. In addition, we used a dataset that consists of all points which have the `wgs84:geometry` property[10] from DBpedia for the comparison with SILK.[11] The 732,224 entities in this dataset are single points on the surface of the planet. We used this dataset because SILK 2.5.3 does not yet support the Hausdorff distance but implements the orthodromic distance.

All experiments were carried out on a 32-core server running JDK 1.7 on Linux 10.04. The processors were 8 quadcore AMD Opteron 6128 clocked at 2.0 GHz. Unless stated otherwise, each experiment was assigned 10GB of memory and was ran 5

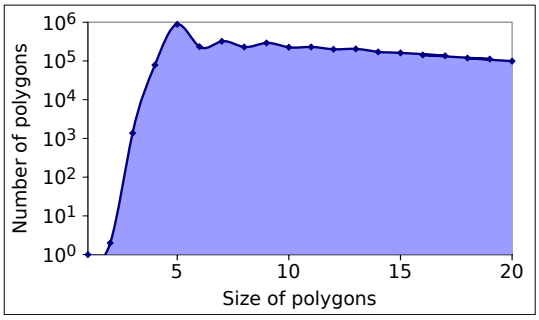
7 We used version 0.9.1 as available at http://nuts.geovocab.org/data/.
8 We used version 3.8 as available at http://dbpedia.org/Datasets.
9 We used the RelevantWays dataset (version of April 26th, 2011) of LinkedGeoData as available at http://linkedgeodata.org/Datasets.
10 wgs84 stands for http://www.w3.org/2003/01/geo/wgs84_pos#.
11 The dataset was extracted from the RelevantNodes dataset (version of April 26th, 2011) of DBpedia as available at http://linkedgeodata.org/Datasets.



(a) Nuts



(b) DBpedia



(c) LGD

Figure 6.3: Distribution of polygon sizes

times. The time-out for experiments was set to 3 hours per iteration. The granularity parameter α was set to 1. In the following, we present the minimal runtime of each of the experiments.

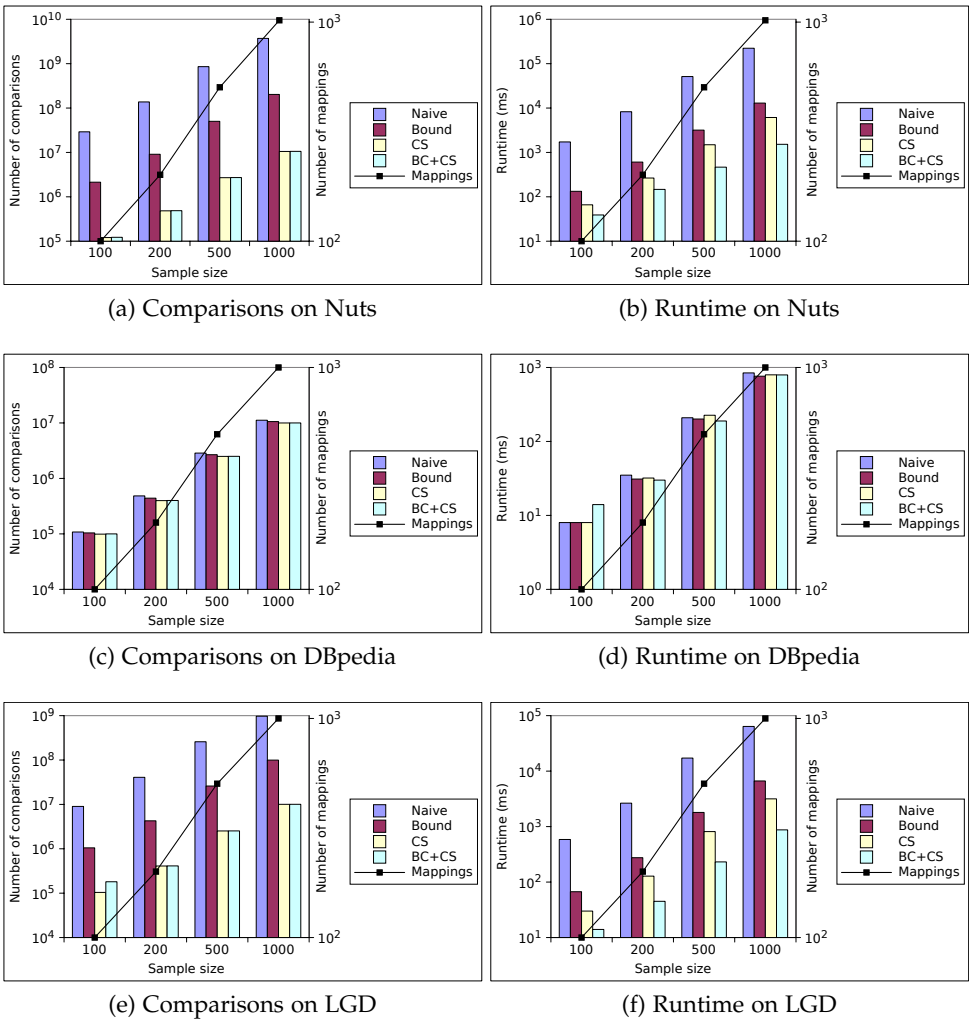
6.5.2 *Results*

6.5.2.1 *Hausdorff Implementations*

In the first part of our evaluation, we measured the runtimes achieved by the three different implementation of the Hausdorff distances on random samples of the Nuts, DBpedia and LGD datasets. We used three different thresholds for our experiments, i.e., 100 m, 0.5 km and 1 km. In Figure 6.4, we present the results achieved with a threshold of 100 m. The results of the same experiments for 0.5 km and 1 km did not provide us with significantly different insights. All exact values can be found on the project website. As expected the runtime of all three approaches increases quadratically with the size of the sample. There is only a slight variation in the number of comparisons (see Figure 6.4) carried by the three approaches on the DBpedia dataset. This is simply due to most polygons in the dataset having only a small number of nodes as shown in Figure 6.3. With respect to runtime, there is no significant difference between the different approaches on DBpedia. This is an important result as it suggests that we can always use the CS or BC + CS approaches even when the complexity of the polygons in the datasets is unknown.

On the two other datasets, the difference between the approaches with respect to both the number of comparisons and the runtime can be seen clearly. Here, the bound implementation requires an order of magnitude less comparisons than the naive approach while the indexed implementations need two orders of magnitude

less comparisons. The runtimes achieved by the approaches reflect the observations achieved on the comparisons. In particular, the bound approach is an order of magnitude faster than the naive approach. Moreover, the BC + CS approach outperforms the bound approach by approximately one further order of magnitude. Note that up to approximately 1.07% of the comparisons carried out by BC + CS are the result of the indexing step.



(a) Comparisons on Nuts (b) Runtime on Nuts

(c) Comparisons on DBpedia (d) Runtime on DBpedia

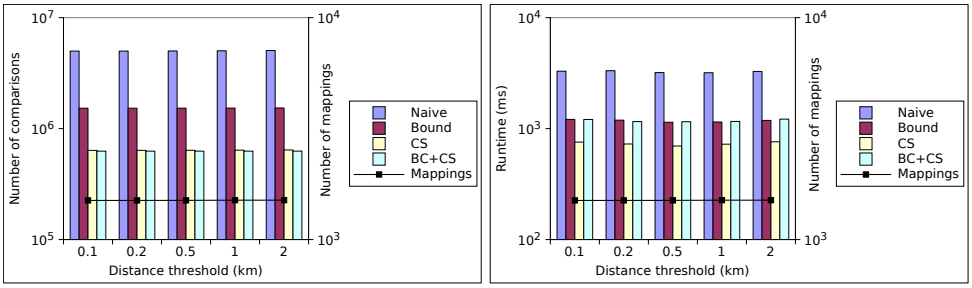
(e) Comparisons on LGD (f) Runtime on LGD

Figure 6.4: Number of comparisons and runtimes on samples of the datasets

6.5.2.2 *Deduplication*

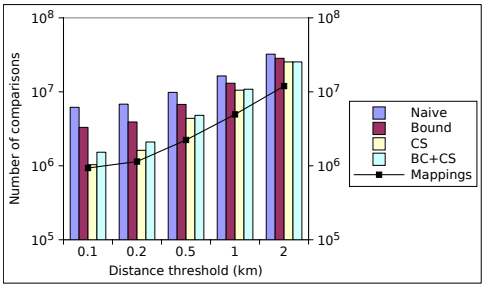
In our second series of experiments, we deduplicated the three datasets at hand by using four different thresholds between 100 m and 2 km. We compared the combination of ORCHID ($\alpha = 1$) and of all different implementations of the Hausdorff distance. The rationale behind this experiment was to measure whether the bound and indexed implementations were of any use even within the smaller subproblems generated by ORCHID. The results achieved show that using these implementations can indeed lead to significant improvements in both runtime and comparisons (see Figure 6.5). In particular, the indexed distance profits from the fact that it can discard a large number of computations that would lead to distance

below and above the distance threshold. Thus, it requires over than two orders of magnitude less computations than the bound and naive versions on the Nuts dataset. Given the small size of the index that it generates for Nuts, the indexed approach is also two orders of magnitude faster across all the thresholds. On the LGD dataset, the indexed approach is the only one that terminated within the set time of 3 hours. Due to the topology of the DBpedia data, the runtimes on DBpedia are comparable for all approaches. Here, it is important to note that for smaller thresholds, the indexed approach still requires close to an order of magnitude less comparisons than the naive approach.

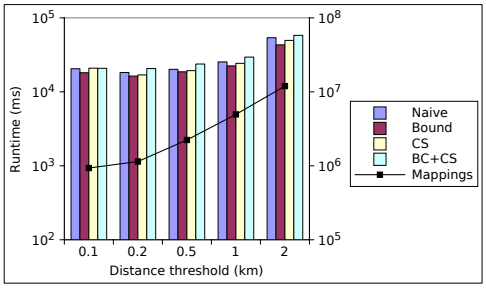


(a) Comparisons on Nuts

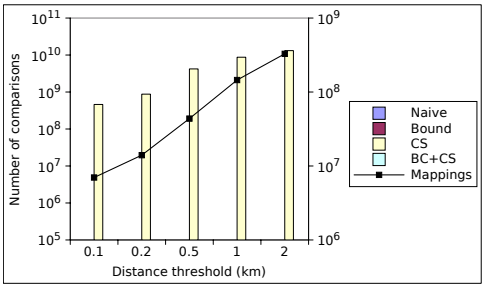
(b) Runtime on Nuts



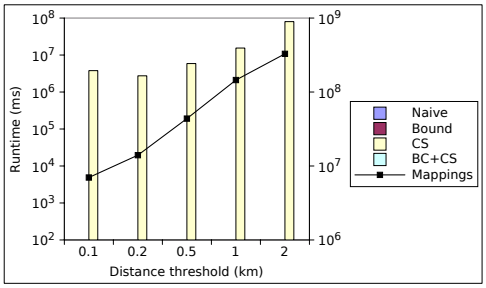
(c) Comparisons on DBpedia



(d) Runtime on DBpedia



(e) Comparisons on LGD



(f) Runtime on LGD

Figure 6.5: Number of comparisons and runtime of ORCHID

6.5.2.3 *Scalability*

We were also interested in knowing how our approach performs with growing dataset sizes. We thus ran ORCHID in combination with BC with randomly selected slices of LinkedGeoData and DBpedia and computed the runtime against the size of the data slices. The similarity threshold was set to 0.1 km as in the previous experiment. The results on DBpedia and LinkedGeoData are shown in Table 6.1. We omitted Nuts because it is too small for scalability experiments. The runtimes and

number of comparisons on DBpedia suggest that the approach behaves in a quasi-linear fashion on low-dimensional and sparsely distributed data. Note that the number of mappings because partly larger than the number of computations on this dataset is simply due to items with the same URI being found in both source and target and thus not necessitating any comparisons for deduplication. This is more rarely the case in the LinkedGeoData dataset. The runtimes on LinkedGeoData yet suggest that both the number of computations and the runtime required of our approach grow sub-linearly with the number of mappings to be computed when the number of points per polygon grows. This can be explained by our approach making effective use of existing data to discard computations and reduce the ratio of number of computations to mappings with growing data size. Thus, our approach promises to scale well to even larger datasets.

Sample Size	od computations	Runtime (ms)	Mappings
10^5	34,959	2,936	103,428
2×10^5	97,798	5,783	215,096
4×10^5	341,986	10,423	459,681
7.3×10^5	1,035,222	20,727	932,848
10^5	5,703,683	42,437	77,003
2×10^5	11,734,609	57,935	159,878
4×10^5	24,844,435	153,174	342,477
8×10^5	55,212,459	411,248	777,826
16×10^5	131,405,064	819,636	1,902,803

Table 6.1: Scalability results. The top section shows the results on DBpedia while the lower section shows the results on LinkedGeoData.

6.5.2.4 *Comparison with other approaches*

SILK[12] (Isele et al., 2011) is of the few other LD framework which implements the orthodromic distance. To the best of our knowledge, no other LD framework implements the Hausdorff distance. Thus, we compare ORCHID in combination with the naive implementation of the Hausdorff distance to SILK on all 732,224 points from DBpedia that contain longitude and latitude information. The results of four different distance thresholds are shown in Figure 6.6. Our results clearly show that ORCHID outperforms SILK by more than one order of magnitude in all settings.

6.6 RELATED WORK

The work presented herein is related to record linkage, deduplication, LD and the efficient computation of Hausdorff distances. An extensive amount of literature has been published by the database community on record linkage (see (Elmagarmid et al., 2007; Köpcke et al., 2010) for surveys). With regard to *time complexity*,

12 Throughout our experiments, we used SILK 2.5.3.

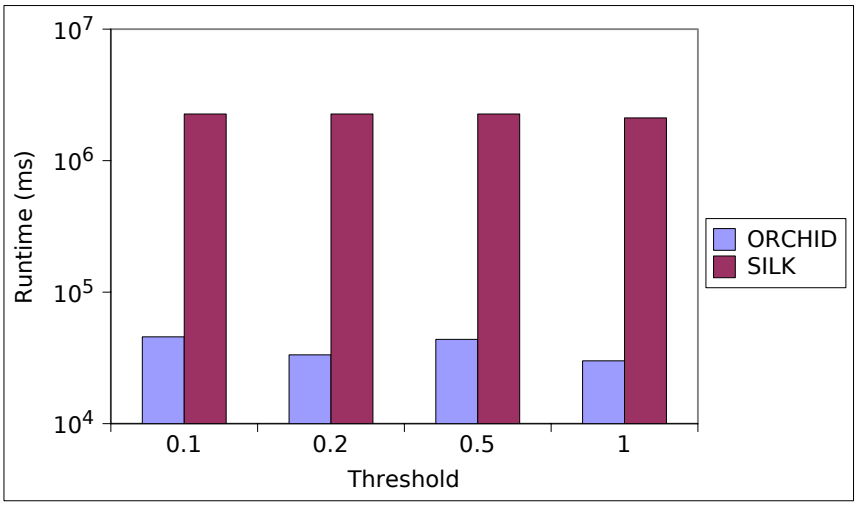


Figure 6.6: Comparison of runtime of SILK and ORCHID

time-efficient deduplication algorithms such as PPJoin+ (Xiao et al., 2008), ED-Join (Xiao et al., 2008), PassJoin (Li et al., 2011) and TrieJoin (Wang et al., 2010; Feng et al., 2012) were developed over the last years. Several of these were then integrated into the hybrid LD framework LIMES (Ngonga Ngomo, 2012b). Moreover, dedicated time-efficient approaches were developed for LD. For example, RDF-AI (Scharffe et al., 2009) implements a five-step approach that comprises the pre-processing, matching, fusion, interlink and post-processing of datasets. (Ngonga Ngomo and Auer, 2011) presents an approach based on the Cauchy-Schwarz that allows discarding a large number of unnecessary computations. The approaches HYPPO (Ngonga Ngomo, 2011) and $\mathcal{HR}^3$ (Ngonga Ngomo, 2012a) rely on space tiling in spaces with measures that can be split into independent measures across the dimensions of the problem at hand. Especially, $\mathcal{HR}^3$ was shown to be the first approach that can achieve a relative reduction ratio r' less or equal to any given relative reduction ratio $r > 1$. Standard blocking approaches were implemented in the first versions of SILK and later replaced with MultiBlock (Isele et al., 2011), a lossless multi-dimensional blocking technique. KnoFuss (Nikolov et al., 2012) also implements blocking techniques to achieve acceptable runtimes.

Hausdorff distances are commonly used in fields such as object modeling, computer vision and object tracking. (Atallah, 1983) presents an approach for the efficient computation of Hausdorff distances between convex polygons. While the approach is quasi-linear in the number of nodes of the polygons, it cannot deal with non-convex polygons as commonly found in geographic data. (Guthe et al., 2005) presents an approach for the comparison of 3D models represented as triangular meshes. The approach is based on a subdivision sampling algorithm that makes use of octrees to approximate the distance between objects. (Tang et al., 2009) present a similar approach that allows approximating Hausdorff distances within a certain error bound while (Bartoň et al., 2010) presents an exact approach. (Nutanong et al., 2011) present an approach to compute Hausdorff distances between trajectories using R-trees within an L_2-space. Note that our approach is tailored to run in orthodromic spaces. Still, some of the insights presented in (Nutanong et al., 2011) may be usable in an orthodromic space. To the best of our knowledge, none of the approaches proposed before address the problem of finding pairs of polygons (A, B) such that $hd(A, B) \leqslant \theta$ in an orthodromic space.

6.7 CONCLUSION AND FUTURE WORK

In this chapter, we presented ORCHID, a LD approach for geographic data. Our approach is based on the combination of Hausdorff and orthodromic distances. We devised two approaches for computing bound Hausdorff distances and compared these approaches with the naive approach. Our experiments showed that we can be more than two orders of magnitude faster on typical geographic datasets such as Nuts and LinkedGeoData. We then presented the space tiling approach which underlies ORCHID and proved that it is reduction-ratio-optimal. Our most interesting result was that our approach seems to be sub-linear with respect to the number of comparisons and the runtime it requires. This behavior can be explained by the approach making use of the higher data density to perform better distance approximations and thus discarding more computations of the orthodromic distance. In addition to comparing different parameter settings of ORCHID with each other, we also compared our approach with the state-of-the-art LD framework SILK. Our results show that we outperform the blocking approach it implements by more than one order of magnitude. In future work, we will extend our approach by implementing it in parallel and integrating it with a load balancing approach.

7

RAPID EXECUTION OF WEIGHTED EDIT DISTANCES

PREAMBLE

This chapter is based on joint work published in (Soru and Ngonga Ngomo, 2013) and addresses the scalable execution of weighted edit distance. The approach followed herein is based on a sequence of filters, which differs from the approaches presented in the previous chapters. The author developed the idea behind this work, supervised the work and co-wrote the paper.

7.1 INTRODUCTION

The computation of string similarities plays a central role in manifold disciplines ranging from computational biology (Bernt et al., 2012) to link discovery on the Web of Data[1] (Soru and Ngonga Ngomo, 2012). Over the last decades, manifold domain-specific string similarities have been developed for improving the accuracy of automatic techniques that rely on them. For example, the Jaro-Winkler similarity was developed especially to perform well on person names (Winkler, 2006). Still, newer works in machine learning have shown that learning string similarities directly from data can lead to algorithms with a performance superior to that of those which rely on standard similarity measures. Especially, work on link discovery on the Web of Data (Soru and Ngonga Ngomo, 2012) has shown that data-specific weighted edit distances can lead to higher F-measures for link specifications.

One main problem has yet plagued the approaches which rely on string similarity measures learned from data: their runtime. While dedicated algorithms for the time-efficient comparison of large volumes of data have been developed over the last years (e.g., PPJoin+ (Xiao et al., 2008), EDJoin (Xiao et al., 2008), PassJoin (Li et al., 2011) and TrieJoin (Feng et al., 2012)), the time-efficient computation of data-specific string similarities has been paid little attention to. Thus, running the data-specific counterparts of standard similarity measures is often orders of magnitude slower. Previous work have circumvented this problem in manifold ways, including the execution of approximations of the data-specific similarity measure. For example, weighted edit distances are sometimes approximated by first computing the edit distance between two strings A and B and only subsequently applying the weight of each of the edit operations (Kurtz, 1996). Other approximations can be found in (Bellet et al., 2011, 2012).

In this paper, we address the problem of the time-efficient computation of weighted edit distances by presenting a novel approach, REEDED. Our approach uses weight bounds on the input cost matrix to efficiently discard similarity computations that would lead to dissimilar pairs. By these means, REEDED can outperform state-of-the-art approaches for the computation of edit distances by more than one order of magnitude on real datasets. We explain our approach on one of its prime ar-

1 Throughout this paper, we use the expression "link discovery" to mean the discovery of typed relations that link instances from knowledge bases on the Web of Data. This discipline is related to entity resolution and deduplication and known from databases.

eas of application (i.e., link discovery (Soru and Ngonga Ngomo, 2012)) by using the data shown in Table 7.1 as example. Here, the task is to detect possible pairs $(s, t) \in S \times T$ such that s `owl:sameAs` t, where S is a set of source resources and T is a set of target resources.

The contributions of our paper can be summarized as follows:

- We present the REEDED approach for the time-efficient computation of weighted edit distances.
- We prove the completeness and correctness of REEDED's results formally.
- We compare REEDED with a weighted version of the state-of-the-art approach PassJoin on 4 datasets and show that we outperform it by more than one order of magnitude.

The rest of this paper is structured as follows: In Section 7.2, we present preliminaries to our work. Thereafter, we give some insights into the intuitions behind our work (Section 7.3). We then present the REEDED approach formally in Section 7.4 and prove that our results are both complete and correct. In Section 7.6, we evaluate our approach on four datasets and show that we outperform the state of the art in all settings. Finally, we conclude with Section 7.8 after giving a brief overview of related work in Section 7.7.

7.2 PRELIMINARIES

7.2.1 *Notation and Problem Statement*

Let Σ be an alphabet and Σ^* be the set all sequences that can be generated by using elements of Σ. We call the elements of Σ characters and assume that Σ contains the empty character ϵ. The edit distance – or Levenshtein distance – of two strings $A \in \Sigma^*$ and $B \in \Sigma^*$ is the minimum number of edit operations that must be performed to transform A into B (Levenshtein, 1965). An *edit operation* can be the insertion or the deletion of a character, or the substitution of a character with another one. In a *plain* edit distance environment, all edit operations have a cost of 1. Thus, the distance between the strings "Generalized" and "Generalised" is the same as the distance between "Diabetes Type I" and "Diabetes Type II". Yet, while the first pair of strings is clearly semantically equivalent for most applications, the elements of the second pair bears related yet significantly different semantics (especially for medical applications). To account for the higher probability of edit operations on certain characters bearing a higher semantic difference, weighted edit distances were developed. In a *weighted* edit distance environment, a cost function $\mathtt{cost} : \Sigma \times \Sigma \rightarrow [0, 1]$ assigned to each of the possible edit operations. The totality all of costs can be encoded in a *cost matrix* M. The cost matrix is quadratic and of dimensions $|\Sigma| \times |\Sigma|$ for which the following holds:

$$\forall i \in \{1, \ldots, |\Sigma|\}\ m_{ii} = 0 \tag{7.1}$$

The entry m_{ij} is the cost for substituting the i^{th} character c_i of Σ with the j^{th} character c_j of the same set. Note that if $c_i = \epsilon$, m_{ij} encode the insertion of c_j. On the other hand, if $c_j = \epsilon$, m_{ij} encode the deletion of c_i.

In most applications which require comparing large sets of strings, string similarities are used to address the following problem: Given a set S of source strings

and a set T of target strings, find the set $\mathcal{R}(S, T, \delta_p, \theta)$ of all pairs $(A, B) \in S \times T$ such that

$$\delta_p(A, B) \leqslant \theta \tag{7.2}$$

where θ is a distance threshold and δ_p is the plain edit distance. Several scalable approaches have been developed to tackle this problem for plain edit distances(see e.g., (Li et al., 2011) and (Xiao et al., 2008)). Still, to the best of our knowledge, no scalable approach has been proposed for finding all $(A, B) \in S \times T$ such that $\delta(A, B) \leqslant \theta$ for weighted edit distances δ. In this paper we address exactly this problem by presenting REEDED. This approach assumes that the computation of weighted edit distances can be carried out by using an extension of the dynamic programming approach used for the plain edit distance.

7.2.2 *Extension of Non-Weighted Approaches*

All of the approaches developed to address the problem at hand with the plain edit distance can be easily extended to deal with weighted edit distances for which the dynamic programming approach underlying the computation of the plain edit distance still holds. Such an extension can be carried out in the following fashion: Let

$$\mu = \min_{0 \leqslant i,j \leqslant |\Sigma|} m_{ij}. \tag{7.3}$$

Then, if the weighted edit distance between two strings A and B is d, then at most d/μ edit operations were carried out to transform A into B. By using this insight, we can postulate that for any weighted edit distance δ with cost matrix M, the following holds

$$\forall A \in \Sigma^* \; \forall B \in \Sigma^* \; \delta(A, B) \leqslant \theta \rightarrow \delta_p(A, B) \leqslant \frac{\theta}{\mu}. \tag{7.4}$$

Thus, we can reduce the task of finding the set $\mathcal{R}(S, T, \delta, \theta)$ to that of first finding $\mathcal{R}(S, T, \delta_p, \theta/\mu)$ and subsequently filtering $\mathcal{R}(S, T, \delta_p, \theta/\mu)$ by using the condition $\delta(A, B) \leqslant \theta$. To the best of our knowledge, PassJoin (Li et al., 2011) is currently the fastest approach for computing $\mathcal{R}(S, T, \delta_p, \theta)$ with plain edit distances. We thus extended it to deal with weighted edit distances and compared it with our approach. Our results show that we outperform the extension of PassJoin by more than one order of magnitude.

7.3 THE REEDED APPROACH

7.3.1 *Overview*

Our approach REEDED (Rapid Execution of Weighted Edit Distances) aims to compute similar strings using weighted edit distance within a practicable amount of time. The REEDED approach is basically composed of three nested filters as shown in Figure 7.1, where each filter takes a set of pairs as input and yields a subset of the input set according to a predefined rule. In the initial step of REEDED, the input data is loaded from S, a source dataset, and T, a target dataset. Their Cartesian product $S \times T$ is the input of the first length-aware filter. The output of the first filter $\mathcal{L}$ is the input of the second character-aware filter. The weighted edit distance will be calculated only for the pairs that pass through the second filter, i.e.

set $\mathcal{N}$. The final result $\mathcal{A}$ is the set of pairs whose weighted edit distance is less or equal than a threshold θ. Note that pairs are processed one by one. This ensures that our algorithm performs well with respect to its space complexity.

Algorithm 7.1 Main algorithm

Require: S, T, p, q, θ
Ensure: $\mathcal{A}$: a set of pairs
1: $\tau \leftarrow \theta / \min(m_{ij})$
2: **for all** $s \in S$ **do**
3: **for all** $t \in T$ **do**
4: **if** $||s| - |t|| \leqslant \tau$ **then**
5: $C_s \leftarrow$ CHARSOF(s)
6: $C_t \leftarrow$ CHARSOF(t)
7: $C \leftarrow C_s \oplus C_t$
8: **if** $\lceil |C|/2 \rceil \leqslant \tau$ **then**
9: $\delta \leftarrow$ WEIGHTEDEDITDISTANCE(s, t)
10: **if** $\delta \leqslant \theta$ **then**
11: $\mathcal{A} \leftarrow \mathcal{A} \cup \langle s, t \rangle$
12: **end if**
13: **end if**
14: **end if**
15: **end for**
16: **end for**
17: **return** $\mathcal{A}$

Algorithm 7.2 Implementation the CHARSOF method

Require: String s
1: $C = \emptyset$
2: **for** $i = 0 \rightarrow |s - 1|$ **do**
3: $C \leftarrow C \cup s[i]$
4: **end for**
5: **return** C

More formally, our algorithm for computing $\mathcal{R}(S, T, \delta, \theta)$ works as described in Algorithm 7.1. The filtering is performed at rows 4, 8, and 10. Note that the functions GETSUBCOST, GETDELCOST and GETINSCOST refer to the costs assigned to every edit operation. These values are stored into the *cost matrix*. In Section 7.6.1 we will explain how the cost matrix has been initialized. In the following, we explain each of the key steps of our algorithm by using the datasets shown in Table 7.1 as example. We will assume the costs $sub(c, C) = sub(t, T) = 0.5$, $ins(s) = 0.6$ and $sub(1, 2) = sub(2, 1) = 0.7$. All other substitutions, deletions and insertions will be assumed to have a cost of 0.

7.3.2 *Key Assumption*

Similarly to the extensions of plain edit distances for weighted edit distances, REEDED assumes the dynamic programming approach commonly used for computing plain edit distances can be used for computing the weighted edit distance

Algorithm 7.3 Implementation the WeightedEditDistance method

Require: Strings a, b
1: $\forall i, j\ L[i,j] \leftarrow 0$
2: **for** $i = 1 \rightarrow |a|$ **do**
3: $L[i,1] \leftarrow i$
4: **end for**
5: **for** $j = 1 \rightarrow |b|$ **do**
6: $L[1,j] \leftarrow j$
7: **end for**
8: **for** $i = 1 \rightarrow |a|$ **do**
9: **for** $j = 1 \rightarrow |b|$ **do**
10: $L[i,j] \leftarrow min\{L[i-1,j-1]+\text{GetSubCost}(a[i],b[j]),$
11: $L[i-1,j]+\text{GetDelCost}(a[i]), L[i,j-1]+\text{GetInsCost}(b[j])\}$
12: **end for**
13: **end for**
14: **return** $L[|a|,|b|]$

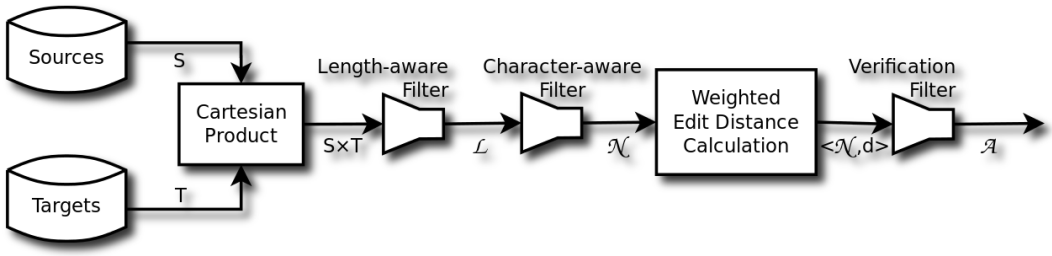


Figure 7.1: Flowchart of the REEDED approach

described by the cost matrix M. With respect to M, this assumption translates to the weights in the matrix being such that there is no sequence of two edit operations m_{ij} and $m_{i'j'}$ that is equivalent to a third edit operation $m_{i''j''}$ with

$$m_{i''j''} > m_{ij} + m_{i'j'}. \tag{7.5}$$

for $(i \neq j) \wedge (i' \neq j') \wedge (i'' \neq j'')$. Formally, we can enforce this condition of the cost matrix M by ensuring that

$$\exists k > 0\ \forall m_{ij} : k < m_{ij} \leqslant 2k. \tag{7.6}$$

Given that the entries in cost matrices are usually bound to have a maximal value of 1, we will assume without restriction of generality that

$$\forall i \in \{1,\ldots,|\Sigma|\} \forall j \in \{1,\ldots,|\Sigma|\}\ i \neq j \rightarrow 0.5 < m_{ij} \leqslant 1. \tag{7.7}$$

Thus, in the following, we will assume that $\forall m_{ij} : m_{ij} > 0.5$. We discuss the case where this assumption is not given in Section 7.5 of the paper.

7.3.3 *Length-aware Filter*

The *length-aware filter* is the first filter of REEDED. Once the datasets have been loaded, the Cartesian product

$$S \times T = \{\langle s,t\rangle : s \in S, t \in T\} \tag{7.8}$$

Sources (S)		Targets (T)	
id	*name*	*id*	*name*
s_1	Basal cell carcinoma	t_1	Basal Cell Carcinoma
s_2	Blepharophimosi	t_2	Blepharophimosis
s_3	Blepharospasm	t_3	Blepharospasm
s_4	Brachydactyly type A1	t_4	Brachydactyly Type A1
s_5	Brachydactyly type A2	t_5	Brachydactyly Type A2

Table 7.1: Example datasets

is computed, which in our example corresponds to $\{\langle s_1, t_1\rangle, \langle s_1, t_2\rangle, \ldots, \langle s_5, t_5\rangle\}$. The basic insight behind the first filter is that given two strings s and t with lengths $|s|$ resp. $|t|$, we need at least $||s| - |t||$ edit operations to transform s into t. Now given that each edit operation costs at least μ, the cost of transforming s to t will be at least $\mu||s| - |t||$. Consequently, the rule which the filter relies on is the following:

$$\langle s, t\rangle \in \mathcal{L} \;\Rightarrow\; \langle s, t\rangle \in S \times T \wedge ||s| - |t|| \leqslant \theta/\mu. \tag{7.9}$$

In the following, we will set $\tau = \theta/\mu$, where μ is as defined in Equation 7.3.

In our example, let us assume $\theta = 1$ and $m_{ij} \in (0.5, 1.0]$. Then, $\tau = 2$. If we assume that *S.name* has been mapped to *T.name*, then at the end of this step, 13 of the 25 initial pairs in $S \times T$ are dismissed. The remaining 8 pairs are:

$$\mathcal{L} = \{\langle s_1, t_1\rangle, \langle s_2, t_2\rangle, \langle s_3, t_3\rangle, \langle s_4, t_4\rangle, \langle s_5, t_5\rangle, \langle s_2, t_3\rangle, \langle s_4, t_5\rangle, \langle s_5, t_4\rangle\}. \tag{7.10}$$

7.3.4 *Character-aware Filter*

The second filter is the *character-aware filter* which only selects the pairs of strings that do not differ by more than a given number of characters. The intuition behind the filter is that given two strings s and t, if $|C|$ is the number of characters that do not belong to both strings, we need at least $\lceil |C|/2 \rceil$ operations to transform s into t. As above, the cost of transforming s to t will be at least $\mu\lceil |C|/2 \rceil$.

The characters of each string are collected into two sets, respectively C_s for the source string and C_t for the target string. Since s and t may contain more than one occurrence of a single character, characters in C_s and C_t are enumerated. Then, the algorithm computes their exclusive disjunction C:

$$C = C_s \oplus C_t. \tag{7.11}$$

Finally, the filter performs the selection by applying the rule:

$$\langle s, t\rangle \in \mathcal{N} \iff \langle s, t\rangle \in \mathcal{L} \wedge \left\lceil \frac{|C|}{2} \right\rceil \leqslant \tau. \tag{7.12}$$

In our example, a further pair can be dismissed by these means, leading to the set of remaining pairs being as follows:

$$\mathcal{N} = \{\langle s_1, t_1\rangle, \langle s_2, t_2\rangle, \langle s_3, t_3\rangle, \langle s_4, t_4\rangle, \langle s_5, t_5\rangle, \langle s_4, t_5\rangle, \langle s_5, t_4\rangle\}$$

The pair that is rejected is $\langle s_2, t_3\rangle$, for which $C = \{h_1, i_1, o_1, i_2, a_1, s_1\}$, which leads to the rule not being satisfied. Note that pair $\langle s_3, t_2\rangle$ could have passed through the filter, but it was not in the length-aware selection.

7.3.5 *Verification*

For all the pairs left in $\mathcal{N}$, the weighted edit distance among is calculated. After that, the third filter selects the pairs whose distance is less or equal than a threshold θ.

$$\langle s,t\rangle \in \mathcal{A} \iff \langle s,t\rangle \in \mathcal{N} \wedge \delta(s,t) \leqslant \theta \tag{7.13}$$

In our example datasets, the set

$$\mathcal{A} = \{\langle s_1,t_1\rangle, \langle s_2,t_2\rangle, \langle s_3,t_3\rangle, \langle s_4,t_4\rangle, \langle s_5,t_5\rangle\} \tag{7.14}$$

is the final result of the selection. Note that the pairs $\langle s_4,t_5\rangle$ and $\langle s_5,t_4\rangle$ are discarded, because their distance (1.2) is greater than the threshold (1.0).

7.4 CORRECTNESS AND COMPLETENESS

In the previous section, we showed that our approach performs as it should on an example. In this section, we prove formally that REEDED does so on any pair of datasets. We achieve this goal by proving that REEDED is both correct and complete, where:

- We say that an approach is *correct* if the output O it returns is such that $O \subseteq \mathcal{R}(S,T,\delta,\theta)$.
- Approaches are said to be *complete* if their output O is a superset of $\mathcal{R}(S,T,\delta,\theta)$, i.e., $O \supseteq \mathcal{R}(S,T,\delta,\theta)$.

As stated in Section 7.3, REEDED consists of three nested filters, each of which creates a subset of pairs.

$$\mathcal{A} \subseteq \mathcal{N} \subseteq \mathcal{L} \subseteq S \times T \tag{7.15}$$

For the purpose of clearness, we name each filtering rule:

$$R_1 \Leftrightarrow ||s| - |t|| \leqslant \tau$$

$$R_2 \Leftrightarrow \left\lceil \frac{|C|}{2} \right\rceil \leqslant \tau$$

$$R_3 \Leftrightarrow \delta(s,t) \leqslant \theta$$

Using Equation 7.9, Equation 7.12 and Equation 7.13, each subset can be redefined as follows:

$$\mathcal{L} = \{\langle s,t\rangle \in S \times T : R_1\} \tag{7.16}$$

$$\mathcal{N} = \{\langle s,t\rangle \in S \times T : R_1 \wedge R_2\} \tag{7.17}$$

$$\mathcal{A} = \{\langle s,t\rangle \in S \times T : R_1 \wedge R_2 \wedge R_3\} \tag{7.18}$$

We then introduce $\mathcal{A}^*$ as the set of pairs whose weighted edit distance is less or equal than the threshold θ.

$$\mathcal{A}^* = \{\langle s,t\rangle \in S \times T : \delta(s,t) \leqslant \theta\} = \{\langle s,t\rangle \in S \times T : R_3\} \tag{7.19}$$

Lemma 1. *The REEDED approach is correct and complete.*

Proof. This lemma is equivalent to showing that $\mathcal{A} = \mathcal{A}^*$. Let us consider all the pairs in $\mathcal{A}$. REEDED's correctness follows directly from Equation 7.18. All the pairs $\langle s, t\rangle \in \mathcal{A}$ satisfy rule R_3, which also defines $\mathcal{A}^*$. Thus $\mathcal{A} \subseteq \mathcal{A}^*$. All we need to show now is REEDED's completeness, i.e., that none of the pairs discarded by the filters actually belongs to $\mathcal{A}^*$.

We are given two strings s and t. Their length differs of $||s| - |t||$ characters, which is also the smallest number of edit operations (all insertions or all deletions) that must be performed to transform s into t. Therefore, the lower bound of their weighted edit distance is the product among the difference and the minimum edit cost:

$$||s| - |t|| \cdot \mu \leqslant \delta(s, t) \tag{7.20}$$

When rule R_3 applies, we have $\delta(s, t) \leqslant \theta$, so from Equation 7.20:

$$||s| - |t|| \leqslant \tau \tag{7.21}$$

which leads to $R_3 \Rightarrow R_1$. Thus, set $\mathcal{A}$ can be rewritten as:

$$\mathcal{A} = \{\langle s, t\rangle \in S \times T : R_2 \wedge R_3\} \tag{7.22}$$

Now, starting from two strings s and t, we call C the exclusive disjunction of their characters.

$$C = C_s \oplus C_t \tag{7.23}$$

so the number of the operations performed to transform s into t is included between $\lceil |C|/2 \rceil$ (all substitutions plus an insertion or a deletion if $|C|$ is odd) and $|C|$ (all insertions or deletions). In other words, the lower bound of their weighted edit distance is

$$\lceil |C|/2 \rceil \, \mu \leqslant \delta(s, t) \tag{7.24}$$

Again, when rule R_3 applies, we have $\delta(s, t) \leqslant \theta$, so from inequality Equation 7.24:

$$\lceil |C|/2 \rceil \leqslant \tau \tag{7.25}$$

which leads to $R_3 \Rightarrow R_2$. Therefore, we rewrite again set $\mathcal{A}$ as:

$$\mathcal{A} = \{\langle s, t\rangle \in S \times T : R_3\} \tag{7.26}$$

which is the definition of $\mathcal{A}^*$ (Equation 7.19). Thus, $\mathcal{A} = \mathcal{A}^*$. □

7.5 EXTENSION TO ALL WEIGHTED EDIT DISTANCES

As already stated in Section 7.3.2, our approach assumes that there is no sequence of two or more operations that leads to the same result as a single operation but is less expensive. This scenario never occurs on plain edit distances, because all the operations have the same cost. Yet, when working with weighted edit distances, it is central to take into account that each operation is equivalent to concatenations (denoted as $\circ$) of operations as expressed in the following equivalence rules:

$$\mathrm{sub}(x, y) \equiv \mathrm{del}(x) \circ \mathrm{ins}(y) \tag{7.27}$$

$$\mathrm{sub}(x, y) \equiv \mathrm{sub}(x, z) \circ \mathrm{sub}(z, y) \tag{7.28}$$

$$\mathrm{ins}(x) \equiv \mathrm{ins}(z) \circ \mathrm{sub}(z, x) \tag{7.29}$$

$$\mathrm{del}(x) \equiv \mathrm{sub}(x, z) \circ \mathrm{del}(z) \tag{7.30}$$

where $x, y, z \in \Sigma$. For example, if $sub(a, b) = 0.9$, $sub(a, c) = 0.1$ and $sub(c, b) = 0.1$, then the sequence of operations $sub(a, b) \circ sub(b, c)$ has to be preferred to $sub(a, b)$. This implies that if the condition expressed in Equation 7.5 does not hold, then each equivalence rule presented above has to be considered during the construction of the Levenshtein matrix L (Algorithm 7.1 line 10). One way of going about addressing this problem is simply by replacing the cost of $sub(x, y)$ with the cost of the cheapest sequence of operations seq that is equivalent to $sub(x, y)$ if seq's cost is smaller than $sub(x, y)$'s cost. This goal can be achieved by implementing a preprocessing of the matrix M which computes all the possible sequences of operations that can potentially lead to $sub(x, y)$ but are less costly. Such a preprocessing can be implemented efficiently by using the A* algorithm with $h(c_i, c_j) = m_{ij}$ as heuristic function. Moreover, the depth of the search tree can be limited to $d = \lfloor 1/\mu \rfloor$.

7.6 EVALUATION

The goal of our evaluation was to quantify how well REEDED performs in comparison to the state of the art. We thus compared REEDED with the extension of PassJoin as proposed in (Li et al., 2011). We chose PassJoin because it was shown to outperform other approaches for the efficient computation of edit distances, including EDJoin (Xiao et al., 2008) and TrieJoin (Feng et al., 2012). Note that we did run an implementation of EdJoin on the DBLP dataset presented below and it required approximately twice the runtime of PassJoin.

7.6.1 *Experimental Setup*

We compared the approaches across several distance thresholds on four different datasets that were extracted from real data (see Table 7.2).[2] The first two of these datasets contained publication data from the datasets DBLP and ACM. The third and fourth dataset contained product labels from the product catalogs Google Products and ABT (Köpcke et al., 2010). We chose these datasets because they were extracted from real data sources and because of the different string length distribution across them. By running our experiments on these datasets, we could thus ensure that our results are not only valid on certain string length distributions. As weight matrix we used a *confusion matrix* built upon the frequency of typographical errors presented in (Kernighan et al., 1990). The original confusion matrices report the number of occurrences f for each error:

$$\Phi^S = \{f_{ij} : \text{substitution of } i \text{ (incorrect) for } j \text{ (correct)}\} \tag{7.31}$$

$$\Phi^I = \{f_{ij} : \text{insertion of } j \text{ after } i\} \tag{7.32}$$

$$\Phi^D = \{f_{ij} : \text{deletion of } j \text{ after } i\} \tag{7.33}$$

For insertion and deletion, we calculate the total frequency:

$$\omega_j^I = \sum_i \Phi_{ij}^I \tag{7.34}$$

$$\omega_j^D = \sum_i \Phi_{ij}^D \tag{7.35}$$

2 The data used for the evaluation is publicly available at http://dbs.uni-leipzig.de/en/research/projects/object_matching/fever/benchmark_datasets_for_entity_resolution.

The weights of our weight matrix are thus defined as:

$$m_{ij} = \begin{cases} 1 - \frac{\Phi^S_{ij}}{2\max(\Phi^S)} & : i \neq \epsilon \wedge j \neq \epsilon \\ 1 - \frac{\omega^I_j - \min(\omega^I)}{2(\max(\omega^I) - \min(\omega^I))} & : i = \epsilon \wedge j \neq \epsilon \\ 1 - \frac{\omega^D_i - \min(\omega^D)}{2(\max(\omega^D) - \min(\omega^D))} & : i \neq \epsilon \wedge j = \epsilon \end{cases} \tag{7.36}$$

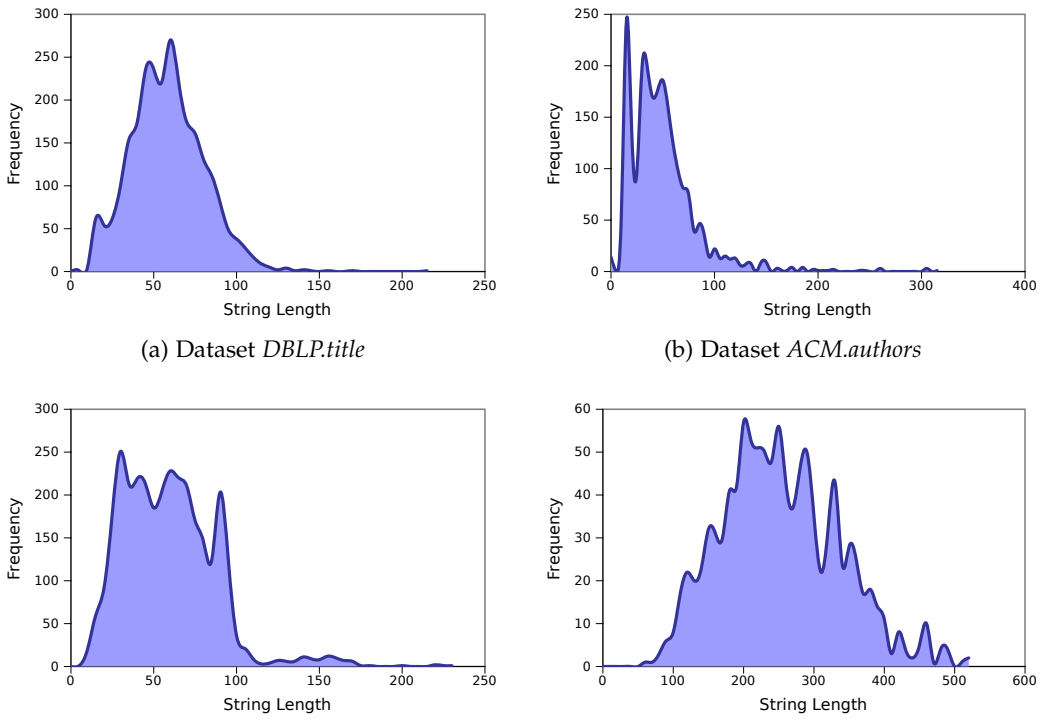
In other words, the higher the probability of an error encoded in m_{ij}, the lower its weight.

Source	Size	Property used	Average string length
DBLP	2,616	Title	56.36
ACM	2,295	Authors	46.64
Google Products	3,226	Product name	57.02
ABT	1,081	Product description	248.18

Table 7.2: Datasets

All experiments were carried out on a 64-bit server running Ubuntu 10.0.4 with 4 GB of RAM and a 2.5 GHz XEON CPU. Each experiment was run 5 times.

7.6.2 *Results*



(a) Dataset *DBLP.title*

(b) Dataset *ACM.authors*

(c) Dataset *GoogleProducts.name*

(d) Dataset *ABT.description*

Figure 7.2: Distribution of string lengths

		PassJoin		REEDED	
Dataset	θ	average	st.dev.	average	st.dev.
DBLP.title	1	10.75	± 0.92	10.38	± 0.35
	2	30.74	± 5.00	15.27	± 0.76
	3	89.60	± 1.16	19.84	± 0.14
	4	246.93	± 3.08	25.91	± 0.29
	5	585.08	± 5.47	37.59	± 0.43
ACM.authors	1	9.07	± 1.05	6.16	± 0.07
	2	18.53	± 0.22	8.54	± 0.29
	3	42.97	± 1.02	12.43	± 0.47
	4	98.86	± 1.98	20.44	± 0.27
	5	231.11	± 2.03	35.13	± 0.35
GoogleProducts.name	1	17.86	± 0.22	15.08	± 2.50
	2	62.31	± 6.30	20.43	± 0.10
	3	172.93	± 1.59	27.99	± 0.19
	4	475.97	± 5.34	42.46	± 0.32
	5	914.60	± 10.47	83.71	± 0.97
ABT.description	1	74.41	± 1.80	24.48	± 0.41
	2	140.73	± 1.40	27.71	± 0.29
	3	217.55	± 7.72	30.61	± 0.34
	4	305.08	± 4.78	34.13	± 0.30
	5	410.72	± 3.36	38.73	± 0.44

Table 7.3: Runtime results in seconds

In Figure 7.2 we show the string length distribution in the datasets. The results of our experiments are shown in Table 7.3. Our results show clearly that REEDED outperforms PassJoin in all experimental settings. On the DBLP dataset (average string length = 56.36), REEDED is already 2 times faster than PassJoin for the threshold 2. For $\theta = 4$, we reach an order of magnitude in difference with runtimes of 25.91 (REEDED) and 246.93 (PassJoin). The runtime of REEDED seems to grow quasi-linearly with increasing values of θ. The results on ACM corroborate the results for the two algorithms. Here, we are 2.16 times faster than PassJoin for $\theta = 2$ and 6.57 times faster for $\theta = 5$. We achieve similar results on the Google Products dataset and are an order of magnitude faster than PassJoin for $\theta = 4$ already. The results we achieve the ABT dataset allow deriving further characteristics of REEDED. Here, the algorithm scales best and requires for $\theta = 5$ solely 1.58 times the runtime it required for $\theta = 1$. This is clearly due to the considerably longer strings contained in this dataset.

We analysed the results on each of the filters in our approach and measure the reduction ratio (given by $1 - |\mathcal{N}|/|S \times T|$) achieved by the length-aware and character-aware filters. Table 7.4 shows the set sizes at each filtering step. Both the first and the second filter reduce the number of selected pairs by one or two orders of magnitude for all the datasets. As expected, the length-aware filter is

DBLP.title	$\theta = 1$	$\theta = 2$	$\theta = 3$	$\theta = 4$	$\theta = 5$
$\lvert S \times T \rvert$	6,843,456	6,843,456	6,843,456	6,843,456	6,843,456
$\lvert \mathcal{L} \rvert$	465,506	832,076	1,196,638	1,551,602	1,901,704
$\lvert \mathcal{N} \rvert$	4,320	4,428	5,726	11,382	30,324
$\lvert \mathcal{A} \rvert$	4,315	4,328	4,344	4,352	4,426
RR(%)	99.94	99.94	99.92	99.83	99.56
ACM.authors	$\theta = 1$	$\theta = 2$	$\theta = 3$	$\theta = 4$	$\theta = 5$
$\lvert S \times T \rvert$	5,262,436	5,262,436	5,262,436	5,262,436	5,262,436
$\lvert \mathcal{L} \rvert$	370,538	646,114	901,264	1,139,574	1,374,482
$\lvert \mathcal{N} \rvert$	3,820	5,070	24,926	104,482	218,226
$\lvert \mathcal{A} \rvert$	3,640	3,708	3,732	3,754	3,946
RR(%)	99.93	99.90	99.53	98.01	95.85
GooglePr.name	$\theta = 1$	$\theta = 2$	$\theta = 3$	$\theta = 4$	$\theta = 5$
$\lvert S \times T \rvert$	10,407,076	10,407,076	10,407,076	10,407,076	10,407,076
$\lvert \mathcal{L} \rvert$	616,968	1,104,644	1,583,148	2,054,284	2,513,802
$\lvert \mathcal{N} \rvert$	4,196	4,720	9,278	38,728	153,402
$\lvert \mathcal{A} \rvert$	4,092	4,153	4,215	4,331	4,495
RR(%)	99.96	99.95	99.91	99.63	95.53
ABT.description	$\theta = 1$	$\theta = 2$	$\theta = 3$	$\theta = 4$	$\theta = 5$
$\lvert S \times T \rvert$	1,168,561	1,168,561	1,168,561	1,168,561	1,168,561
$\lvert \mathcal{L} \rvert$	22,145	38,879	55,297	72,031	88,299
$\lvert \mathcal{N} \rvert$	1,131	1,193	1,247	1,319	1,457
$\lvert \mathcal{A} \rvert$	1,087	1,125	1,135	1,173	1,189
RR(%)	99.90	99.90	99.89	99.88	99.87

Table 7.4: Number of pairs (s, t) returned by each filter. RR stands for the reduction ratio achieved by the combination of length-aware and character-aware filters.

most effective on datasets with large average string lengths. For example, only 1.9% of the Cartesian product of the ABT dataset makes it through the first filter for $\theta = 1$ while the filter allows 6.8% of the DBLP Cartesian product through for $\theta = 1$. One interesting characteristic of the approach is that the size of the $\mathcal{L}$ grows quasi linearly with the value of θ. The character-aware filter seems to have the opposite behavior to the length-aware filter and can discard more string pair on data with small average string lengths. For example, less than 1% of $\mathcal{L}$ makes it through the filter for $\theta = 1$ on the DBLP dataset while 5.1% of $\mathcal{L}$ makes it through the same filter for $\theta = 1$ on ABT.

We also measured the runtime improvement as well as the precision and recall we achieved by combining REEDED with the ACIDS approach and applying this combination to the datasets reported in (Soru and Ngonga Ngomo, 2012). The results are shown in Table 7.5. For the datasets on which the edit distance can be used, the approach achieves a superior precision and recall than state-of-the-art approaches (such as MARLIN (Bilenko and Mooney, 2003) and Febrl (Christen,

	DBLP–ACM		DBLP–Scholar		ABT–Buy	
Labeled examples	20	40	20	40	20	40
F-score (%)	88.98	97.92	70.22	87.85	0.40	0.60
Precision (%)	96.71	96.87	64.73	91.88	0.20	0.30
Recall (%)	82.40	99.00	76.72	84.16	100.00	100.00
Without REEDED	27,108	26,316	30,420	30,096	44,172	43,236
With REEDED	14.25	14.24	668.62	668.62	13.03	65.21

Table 7.5: Results for the combination of ACIDS and REEDED. The runtimes in the 2 rows at the bottom are in seconds.

2008)) which do not rely on data-specific measures. Yet, on more noisy datasets, the approach leads to poorer results. In particular, the edit distance has been shown not to be a good measure when the strings to be compared are too long. Also, the words contained in the source string may be completely different from the words contained in the target string, yet referring to the same meaning. A notable shortcoming of the ACIDS approach is the runtime, wherein the learning system iterated for at least 7 hours to find the weight configuration of the weighted edit distance and optimize the classification (Soru and Ngonga Ngomo, 2012). As shown in Table 7.5, REEDED enhances the execution time of ACIDS reducing the total runtime by 3 orders of magnitude on the DBLP–ACM and the ABT–Buy dataset.

7.7 RELATED WORK

Our work is mostly related to the rapid execution of similarity functions and link discovery. Time-efficient string comparison algorithms such as PPJoin+ (Xiao et al., 2008), EDJoin (Xiao et al., 2008), PassJoin (Li et al., 2011) and TrieJoin (Feng et al., 2012) were developed for the purpose of entity resolution and were integrated into frameworks such as *LIMES* (Ngonga Ngomo, 2012b). In addition to time-efficient string similarity computation approaches for entity resolution, approaches for the efficient computation string and numeric similarities were developed in the area of link discovery. For example, (Ngonga Ngomo and Auer, 2011) presents an approach based on the Cauchy-Schwarz inequality. The approaches HYPPO (Ngonga Ngomo, 2011) and $\mathcal{HR}^3$ (Ngonga Ngomo, 2012a) rely on space tiling in spaces with measures that can be split into independent measures across the dimensions of the problem at hand. Especially, $\mathcal{HR}^3$ was shown to be the first approach that can achieve a relative reduction ratio r' less or equal to any given relative reduction ratio $r > 1$. Another way to go about computing $\mathcal{R}(S, T, \delta, \theta)$ lies in the use of lossless blocking approaches such MultiBlock (Isele et al., 2011).

Manifold approaches have been developed on string similarity learning (see, e.g., (Bellet et al., 2011, 2012; Bilenko and Mooney, 2003; Ristad and Yianilos, 1998)). (Bilenko and Mooney, 2003) for example learns edit distances by employing batch learning and SVMs to record deduplication and points out that domain-specific similarities can improve the quality of classifiers. (Bellet et al., 2011, 2012) rely on a theory for good edit distances developed by (Balcan et al., 2008) to determine classifiers based on edit distances that are guaranteed to remain under a given

classification error. Yet, to the best of our knowledge, REEDED is the first approach for the time-efficient execution of weighted edit distances.

7.8 CONCLUSION

In this paper we presented REEDED, an approach for the time-efficient comparison of sets using weighted distances. After presenting the intuitions behind our approach, we proved that it is both correct and complete. We compared our approach with an extension of PassJoin for weighted edit distances and showed that we are more than an order of magnitude faster on 4 different datasets. REEDED is the cornerstone of a larger research agenda. As it enable to now run weighted edit distances on large datasets within acceptable times, it is also the key to developing active learning systems for link discovery that do not only learn link specifications but also similarity measures directly out of the data. As shown in (Soru and Ngonga Ngomo, 2012), this combination promises to outperform the state of the art, which has relied on standard measures so far. In future work, we will thus combine REEDED with specification learning approaches such as EAGLE (Ngonga Ngomo and Lyko, 2012) and RAVEN (Ngonga Ngomo et al., 2011) and study the effect of weighted edit distances on these approaches.

8 EXECUTION OPTIMIZATION FOR LINK DISCOVERY

PREAMBLE

So far, we presented approaches that allow execution single measures rapidly. The aim of this chapter is to optimize the execution of whole link specifications. The contents of the chapter are from (Ngonga Ngomo, 2014), to which the author was the sole contributor.

8.1 INTRODUCTION

Link Discovery (LD) plays a central role in the realization of the Linked Data paradigm. Several frameworks such as LIMES (Ngonga Ngomo, 2012b) and SILK (Isele et al., 2011) have been developed to address the time-efficient discovery of links. These frameworks take a *link specification* (short: LS, also called linkage rule (Isele et al., 2011)) as input. Each LS is converted internally into a sequence of operations which is then executed. While relying on time-efficient algorithms (e.g., PPJoin+ (Xiao et al., 2008) and $\mathcal{HR}^3$ (Ngonga Ngomo, 2012a)) for single operations has been shown to be very time-efficient (Ngonga Ngomo, 2012b), the optimization of the execution of whole LS within this paradigm has been payed little attention to.

In this paper, we address this problem by presenting HELIOS, the (to the best of our knowledge) first execution optimizer for LD. HELIOS aims to reduce the costs necessary to execute a LS. To achieve this goal, our approach relies on two main components: a *rewriter* and a *planner*. The rewriter relies on algebraic operations to transform an input specification into an equivalent specification deemed less time-consuming to execute. The planner maps specifications to *execution plans*, which are sequences of operations from which a mapping results. HELIOS' planner relies on time-efficient evaluation functions to generate possible plans, approximate their runtime and return the one that is likely to be most time-efficient.[1] Our contributions are:

1. We present a novel generic representation of LS as bi-partite trees.
2. We introduce a novel approach to rewriting LS efficiently.
3. We explicate a novel planning algorithm for LS.
4. We evaluate HELIOS on 2097 LS (17 manually and 2080 automatically generated) and show that it outperforms the state of the art by up to two orders of magnitude.

The rest of this paper is structured as follows: First, we present a formal specification of LS and execution plans for LS. Then, we present HELIOS and its two main

1 HELIOS was implemented in the LIMES framework. All information to the tool can be found at http://limes.sf.net. A graphical user interface for the tool can be accessed via the SAIM interface at http://aksw.org/projects/SAIM.

components. Then, we evaluate HELIOS against the state of the art. Finally, we give a brief overview of related work and conclude.

8.2 FORMAL SPECIFICATION

In the following, we present a graph grammar for LS. We employ this grammar to define a *normal form* (NF) for LS that will build the basis for the rewriter and planner of HELIOS. Thereafter, we present execution plans for LS, which formalize the sequence of operations carried out by execution engines to generate links out of specifications. As example, we use the RDF graphs shown in Listing 8.1 and Listing 8.2, for which the perfect LD result set is {(`ex1:P1`, `ex2:P1`), (`ex1:P2`, `ex2:P2`), (`ex1:P3`, `ex2:P3`), (`ex1:P4`, `ex2:P4`)}.

8.2.1 *Normal Form for Link Specifications*

Formally, most LD tools aim to discover the set $\{(s,t) \in S \times T : R(s,t)\}$ provided an input relation R (e.g., `owl:sameAs`), a set S of source resources (for example descriptions of persons) and a set T of target resources. To achieve this goal, declarative LD frameworks rely on LS, which describe the conditions under which $R(s,t)$ can be assumed to hold for a pair $(s,t) \in S \times T$. Several grammars have been used for describing LS in previous works (Isele et al., 2011; Ngonga Ngomo, 2012a; Nikolov et al., 2012). In general, these grammars assume that LS consist of two types of atomic components: *similarity measures* m, which allow comparing property values of input resources and *operators* op, which can be used to combine these similarities to more complex specifications.

Listing 8.1: Example graph 1

```
ex1:P1 ex:label "Anna"@en .
ex1:P1 ex:age "12"^^xsd:integer .
ex1:P1 a ex:Person .
ex1:P2 ex:label "Jack"@en .
ex1:P2 ex:age "15"^^xsd:integer .
ex1:P2 a ex:Person .
ex1:P3 ex:label "John"@en .
ex1:P3 ex:age "16"^^xsd:integer .
ex1:P3 a ex:Person .
ex1:P4 ex:label "John"@en .
ex1:P4 ex:age "19"^^xsd:integer .
ex1:P4 a ex:Person .
```

Listing 8.2: Example graph 2

```
ex2:P1 ex:label "Ana"@en .
ex2:P1 ex:age "12"^^xsd:integer .
ex2:P1 a ex:Person .
ex2:P2 ex:label "Jack"@en .
ex2:P2 ex:age "14"^^xsd:integer .
ex2:P2 a ex:Person .
ex2:P3 ex:label "Joe"@en .
ex2:P3 ex:age "16"^^xsd:integer .
ex2:P3 a ex:Person .
```

```
ex2:P4 ex:label "John"@en .
ex2:P4 ex:age "19"^^xsd:integer .
ex2:P4 a ex:Person .
```

Without loss of generality, a similarity measure m can be defined as a function $m : S \times T \rightarrow [0, 1]$. We use *mappings* $M \subseteq S \times T \times [0, 1]$ to store the results of the application of a similarity function to $S \times T$ or subsets thereof. We also store the results of whole link specifications in mappings. The set of all mappings is denoted by $\mathcal{M}$. We call a measure *atomic* iff it relies on exactly one similarity measure σ (e.g., the edit similarity, dubbed `edit`)[2] to compute the similarity of a pair $(s, t) \in S \times T$ with respect to the (list of) properties p_s of s and p_t of t and write $m = \sigma(p_s, p_t)$. A similarity measure m is either an atomic similarity measure or the combination of two similarity measures via a *metric operator* such as max, min or linear combinations. For example, `edit`(s.label, t.label) is an atomic measure while max(`edit`(s.label, t.label), `edit`(s.age, t.age)) is a complex similarity measure.

We define a *filter* as any function which maps a mapping M to another mapping M'. *Similarity filters* $f(m, \theta)$ return $f(m, \theta, M) = \{(s, t, r') | \exists r : (s, t, r) \in M \wedge m(s, t) \geqslant \theta \wedge r' = \min\{m(s, t), r\}\}$. *Threshold filters* $i(\theta)$ return $i(\theta, M) = \{(s, t, r) \in M : r \geqslant \theta\}$. Note that $i(0, M) = M$ and that we sometimes omit M from similarity filters for the sake of legibility.

We call a specification *atomic* when it consists of exactly one filtering function. For example, applying the atomic specification $f(\text{edit}(\texttt{ex:label}, \texttt{ex:label}), 1)$ to our input data leads to the mapping {(`ex1:P3`, `ex2:P4`, 1), (`ex1:P2`, `ex2:P2`, 1), (`ex1:P4`, `ex2:P4`, 1)}. A complex specification can be obtained by combining two specifications L_1 and L_2 by (1) a *mapping operator* (that allows merging the mappings which result from L_1 and L_2) and (2) a subsequent filter that allows post-processing the results of the merging.[3] In the following, we limit ourselves to the operators based on $\cup$, $\cap$ and $\setminus$ (set difference), as they are sufficient to describe any operator based on set operators. We extend these operators to mappings as follows:

- $M_1 \cap M_2 = \{(s, t, r) : \exists a, b\ (s, t, a) \in M_1 \wedge (s, t, b) \in M_2 \wedge r = \min(a, b)\}$.
- $M_1 \cup M_2 = \{(s, t, r) : (\neg\exists (s, t, a) \in M_1 \wedge (s, t, r) \in M_2) \vee (\neg\exists (s, t, b) \in M_2 \wedge (s, t, r) \in M_1) \vee (\exists (s, t, a) \in M_1 \wedge \exists (s, t, b) \in M_2 \wedge r = \max(a, b))\}$.
- $M_1 \setminus M_2 = \{(s, t, r) \in M_1 : \neg\exists (s, t, a) \in M_2\}$.

For example, if M_1 = {(`ex1:P1`, `ex2:P2`, 1), (`ex1:P1`, `ex2:P3`, 1)} and M_2 = {(`ex1:P1`, `ex2:P2`, 0.5)} then $M_1 \cup M_2 = M_1$, $M_1 \cap M_2 = M_2$ and $M_1 \setminus M_2$ = {(`ex1:P1`, `ex2:P3`, 1)}.

Based on this grammar, we can regard all LS as *bi-partite directed trees* $L = (V(L), E(L))$ which abide by the following restrictions:

1. The vertices of L can be either *filter nodes* $f \in F$ or *operator nodes* $op \in OP$, i.e., $V(L) = F \cup OP$. The leaves and the root of L are always filter nodes. The leaves are filters that run on $S \times T$.

2 We define the edit similarity of two strings s and t as $(1 + lev(s, t))^{-1}$, where lev stands for the Levenshtein distance.

3 We rely on binary operators throughout this paper because n-ary set operators can always be mapped to a sequence of binary operators.

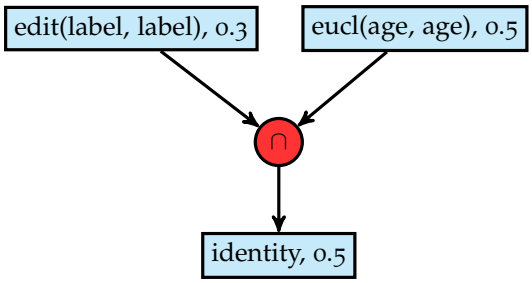


Figure 8.1: A link specification for linking the datasets Person1 and Person2. The filter nodes are rectangles while the operator nodes are circles. $\texttt{eucl}(s.age, t.age) = (1 + |s.age - t.age|)^{-1}$.

2. Edges in L can only exist between filters and operators, i.e., $E(L) \subseteq (F \times OP) \cup (OP \times F)$.

An example of a LS is shown in Figure 8.1. We call this representation of LS their NF. In the rest of this paper, we deal exclusively with finite specifications, i.e., specifications such that their NF contains a finite number of nodes. We call the number of filter nodes of a specification L the *size* of L and denote it $|L|$. For example, the size of the specification in Figure 8.1 is 3. We dub the direct child of L's root the *operator of* L. For example, the operator of the specification in Figure 8.1 is $\cap$. We call a LS L' a *sub-specification* of L (denoted $L' \subseteq L$) if L''s NF is a sub-tree of L's NF that abides by the definition of a specification (i.e., if the root of L''s NF is a filter node and the NF of L' contains all children of L' in L). For example, $f(\texttt{edit}(label, label), 0.3)$ is a sub-specification of our example. We call a L' a *direct sub-specification* of L (denoted $L' \subset_1 L$) if L' is a sub-specification of L whose root node a grandchild of the L's root. For example, $f(\texttt{edit}(label, label), 0.3)$ is a direct sub-specification of the LS shown in Figure 8.1. Finally, we transliterate LS by writing $f(m, \theta, op(L_1, L_2))$ where $f(m, \theta)$ is L's root, op is L's operator, $L_1 \subset_1 L$ and $L_2 \subset_1 L$.

8.2.2 *Execution Plans*

We define an execution plan P as a sequence of *processing steps* $p_1, ..., p_n$ of which each is drawn from the set $\mathcal{A} \times \aleph \times \mathcal{T} \times \mathcal{M} \times \mathcal{M}$, where:

1. $\mathcal{A}$ is the set of all *actions* that can be carried out. This set models all the processing operations that can be carried out when executing a plan. These are:
   a) `run`, which runs the computation of filters $f(m, \theta)$ where m is an atomic measure. This action can make use of time-efficient algorithms such as $\mathcal{HR}^3$.
   b) `filter`, which runs filters $f(m, \theta)$ where m is a complex measure.
   c) `filterout`, which runs the negation of $f(m, \theta)$.
   d) Mapping operations such as `union`, `intersection` and `minus` (mapping difference) and
   e) `return`, which terminates the execution and returns the final mapping.

   The result of each action (and therewith of each processing step) is a mapping.

Canonical Plan	**Abbreviated Canonical Plan**
M_1=(run,edit(label,label),0.3,$\emptyset$,$\emptyset$)	M_1=(run,edit(label,label),0.3)
M_2=(run,eucl(age,age),0.5,$\emptyset$,$\emptyset$)	M_2=(run,eucl(age,age),0.5)
M_3=(intersection,$\emptyset$,$\emptyset$,M_1,M_2)	M_3=(intersection,M_1,M_2)
M_4=(return,$\emptyset$,$\emptyset$,M_3,$\emptyset$)	M_4=(return,M_3)
Alternative Plan1 (abbreviated)	**Alternative Plan2** (abbreviated)
M_1=(run,edit(label,label),0.3)	M_1=(run,eucl(age,age),0.5)
M_2=(filter,eucl(age,age),0.5,M_1)	M_2=(filter,edit(label,label),0.3,M_1)
M_3=(return,M_2)	M_3=(return,M_2)

Table 8.1: Plans for the specification shown in Figure 8.1

2. $\aleph$ is the set of all complex measures as described above united with the $\emptyset$-measure, which is used by actions that do not require measures (e.g., `return`).
3. $\mathcal{T}$ is the set of all possible thresholds (generally $[0, 1]$) united with the $\emptyset$-threshold for actions that do not require any threshold (e.g., `union`) and
4. $\mathcal{M}$ is the set of all possible mappings, i.e., the powerset of $S \times T \times [0, 1]$.

We call the plan P atomic if it consists of exactly one processing step. An *execution planner* EP is a function which maps a LS to an execution plan P. The *canonical planner* EP_0 is the planner that runs specification in *postorder*, i.e., by traversing the NF of LS in the order left-right-root. The approach currently implemented by LIMES (Ngonga Ngomo, 2012b) is equivalent to EP_0. For example, the plan generated by EP_0 for Figure 8.1 is shown in the left column of Table 8.1. For the sake of brevity and better legibility, we will use abbreviated versions of plans that do not contain $\emptyset$ symbols. The abbreviated version of the plan generated by EP_0 for the specification in Figure 8.1 is shown in the right column of Table 8.1. We call two plans *equivalent* when they return the same results for all possible S and T. We call a planner *complete* when it always returns plans that are equivalent to those generated by EP_0.

The insight behind our paper is that equivalent plans can differ significantly with respect to their runtime. For example, the canonical plan shown in Table 8.1 would lead to 32 similarity computations (16 for `edit` and 16 for `euclidean`) and one mapping intersection, which can be computed by using 16 lookups. If we assume that each operation requires 1ms, the total runtime of this plan would be 48ms. The alternative plan 1 shown in Table 8.1 is equivalent to the plans in Table 8.1 but only runs 16 computations of `edit` (leading to M_1 of size 6) and 6 computations of `euclidean` on the data contained in M_1. The total runtime of this plan would thus be 22ms. Detecting such *runtime-efficient* and *complete plans* is the goal of HELIOS.

8.3 HELIOS

HELIOS is an optimizer for LS which consists of two main components: a rewriter (denoted RW) and a planner (denoted HP). Each LS L to be processed is first forwarded to RW, which applies several algebraic transformation rules to transform

L into an equivalent LS L′ that promises to be more efficient to execute. The aim of HP is then to derive a *complete plan* P for L′. This plan is finally sent to the execution engine, which runs the plan and returns a final mapping. In the following, we present each of these components.[4] Throughout the formalization, we use $\rightarrow$ for logical implications and $\Rightarrow$ to denote rules.

8.3.1 *The HELIOS Rewriter*

RW implements an iterative rule-based approach to rewriting. Each iteration consists of three main steps that are carried out from leaves towards the root of the input specification. In the first step, sub-graphs of the input specification L are replaced with equivalent sub-graphs which are likely to be more efficient to run. In a second step, dependency between nodes in L are determined and propagated. The third step consists of removing portions of L which do not affect the final results of L's execution. These three steps are iterated until a fixpoint is reached.

8.3.1.1 *Step 1: Rewriting*

Given a LS L, RW begins by rewriting the specification using algebraic rules dubbed *leaf generation rules*.

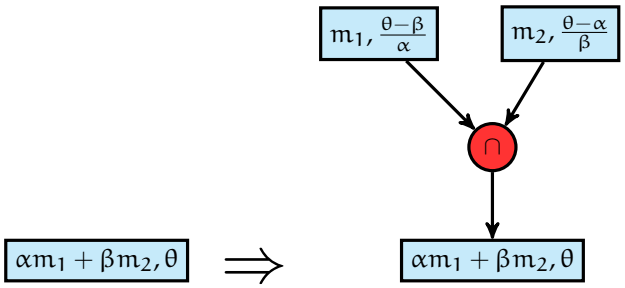


Figure 8.2: Leaf generation rule for linear combinations

The leaf generation rules ($\mathcal{LR}$) make use of relations between metric operators and specification operators to transform leaf nodes with complex measures into graphs whose leaves contain exclusively atomic measures. For example, the rule shown in Figure 8.2 transforms a filter that relies on the linear combinations of 2 measures into a LS with three filters whose leaves only contain atomic measures as described in (Ngonga Ngomo, 2012b). While it might seem absurd to alter the original filter in this manner, the idea here is that we can now run specialized algorithms for m_1 and m_2, then compute the intersection M of the resulting mapping and finally simply check each of the (s, t) with $\exists r : (s, t, r) \in M$ for whether it abides by the linear combination in the root filter. This approach is usually more time-efficient than checking each $(s, t) \in S \times T$ for whether it abides by the linear combination in the original specification. Similar rules can be devised for min (see Figure 8.3), max (see Figure 8.4)and the different average functions used in LD frameworks. After L has been rewritten by the rules in $\mathcal{LR}$, each of its leaves is a filter with atomic measures.

4 Due to space restrictions, some of the details and proofs pertaining to the rewriter and planner are omitted. Please consult http://limes.sf.net for more details.

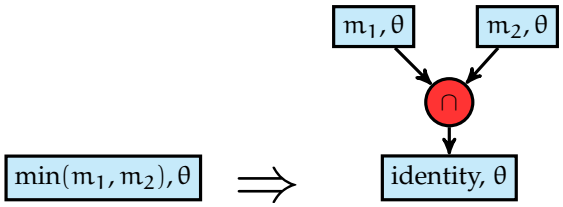


Figure 8.3: Rule for minimum. In the corresponding rule for maximum, the mapping union is used.

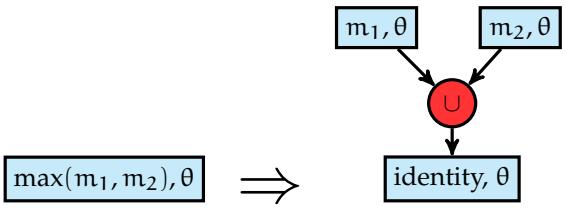


Figure 8.4: Rule for maximum

8.3.1.2 *Step 2: Dependency Detection and Propagation*

The idea behind the use of *dependencies* is to detect and eliminate redundant portions of the specification. Consequently, RW implements two types of dependency-based rules: *dependency detection rules* and *dependency propagation rules*. Formally, we say that L_1 *depends* on L_2 (denoted $\mathtt{depends}(L_1, L_2)$) if the mapping resulting from L_1 is a subset of the mapping resulting from L_2 for all possible S and T. RW generates dependencies between leaves (which now only contain atomic measures) by making use of

$$L_1 = f(m, \theta_1) \wedge L_2 = f(m, \theta_2) \wedge \theta_1 \geqslant \theta_2 \Rightarrow \mathtt{depends}(L_1, L_2). \tag{8.1}$$

Moreover, RW makes use of dependencies have been shown to apply between several similarity and distance measures that are commonly used in literature. For example,the authors of (Xiao et al., 2008) show that for two non-empty strings x and y, $\mathtt{jaccard}(x, y) \geqslant \theta \rightarrow \mathtt{overlap}(x, y) \geqslant \frac{\theta}{1+\theta}(|x| + |y|)$. Given that $|x| \geqslant 1$ and $|y| \geqslant 1$, we can infer that

$$\mathtt{jaccard}(x, y) \geqslant \theta \rightarrow \mathtt{overlap}(x, y) \geqslant \frac{2\theta}{1+\theta}. \tag{8.2}$$

Thus, if $L_1 = f(\mathtt{jaccard}(p_s, p_t), \theta_1)$ and $L_2 = f(\mathtt{overlap}(p_s, p_t), \theta_2)$ with $\theta_2 \leqslant \frac{2\theta_1}{1+\theta_1}$, then $\mathtt{depends}(L_1, L_2)$ holds. Currently, RW implements dependencies between the `overlap`, `trigrams` and the `jaccard` similarities discussed in (Xiao et al., 2008).

Leaf-level dependencies can be propagated towards the root of the specification based on the following rules:

p_1: $L = i(\theta, op(L_1, L_2)) \wedge L_1 = f(m, \theta_1, op_1(L_{11}, L_{12})) \wedge L_2 = f(m, \theta_2, op_2(L_{21}, L_{22})) \wedge \theta_1 \geqslant \theta \wedge \theta_2 \geqslant \theta) \Rightarrow L := i(0, op(L_1, L_2))$ (if the threshold of the father of any operator is smaller than that of all its children and the father node is an identity filter, then the threshold of the father can be set to 0).

p_2: $\mathtt{depends}(L_1, L') \wedge \mathtt{depends}(L_2, L') \wedge L = f(m, \theta, \cap(L_1, L_2)) \Rightarrow \mathtt{depends}(L, L')$ (if all children of a conjunction depend on L' then the father of this conjunction depends on L').

p_3: $L = f(m, 0, \cup(L_1, L_2)) \wedge (depends(L', L_1) \vee depends(L', L_2)) \Rightarrow depends(L', L)$ (if L' depends on one child of a disjunction and the father of the disjunction has the threshold o then L' depends on the father of the disjunction).

8.3.1.3 *Step 3: Reduction*

Given two specifications $L_1 \subset_1 L$ and $L_2 \subset_1 L$ with $depends(L_1, L_2)$, we can now *reduce* the size of $L = filter(m, \theta, op(L_1, L_2))$ by using the following rules:

r_1: $L' = filter(m, \theta, \cap(L_1, L_2)) \wedge depends(L_1, L_2) \Rightarrow L' := filter(m, \theta, L_1))$,

r_2: $L' = filter(m, \theta, \cup(L_1, L_2)) \wedge depends(L_1, L_2) \Rightarrow L' := filter(m, \theta, L_2))$,

r_3: $L' = filter(m, \theta, \setminus(L_1, L_2)) \wedge depends(L_1, L_2) \Rightarrow L' := \emptyset$ where $:=$ stands for overwriting.

An example that elucidates the ideas behind $\mathcal{DR}$ is given in Figure 8.5. Set operators applied to one mapping are assumed to not alter the mapping.

The leaf generation terminates after at most as many iterations as the total number of atomic specifications used across all leaves of the input LS L. Consequently, this step has a complexity of $O(|L'|)$ where $L' = \mathcal{LR}(L)$. The generation of dependencies requires $O(|L'|^2)$ node comparisons. Each time a reduction rule is applied, the size of the L' decreases, leading to reduction rules being applicable at most $|L'|$ times. The complexity of the reduction is thus also $O(|L'|)$. In the worst case of a left- or right-linear specification, the propagation of dependencies can reach the complexity $O(|L'|^2)$. All three steps of each iteration thus have a complexity of at most $O(|L|'^2)$ and the specification is at least one node smaller after each iteration. Consequently, the worst-case complexity of the rewriter is $O(|L'|^3)$.

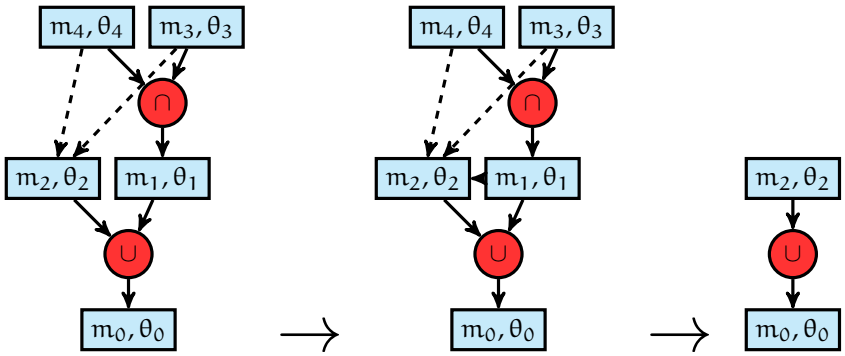


Figure 8.5: Example of propagation of dependencies. The dashed arrows represent dependencies. The dependencies from the left figure are first (using rule p_1). Then, the reduction rule r_2 is carried out, leading to the specification on the right.

8.3.2 *The HELIOS Planner*

The goal of the Helios planner HP is to convert a given LS into a plan. Previous work on query optimization for databases have shown that finding the optimal plan for a given query is exponential in complexity (Silberschatz et al., 2006). The complexity of finding the perfect plan for a LS is clearly similar to that of finding a play for a given query. To circumvent the complexity problem, we

rely on the following optimality assumption: Given $L_1 \subset_1 L$ and $L_2 \subset_1 L$ with $L = f(m, \theta, op(L_1, L_2))$, a good plan for L can be derived from plans for L_1 and L_2. In the following, we begin by explaining core values that HP needs to evaluate a plan. In particular, we explain how HP evaluates atomic and complex plans. Thereafter, we present the algorithm behind HP and analyze its complexity.

8.3.2.1 *Plan Evaluation*

HP uses two values to characterize any plan P: (1) the approximate runtime of P (denoted $\gamma(P)$) and (2) the selectivity of P (dubbed $s(P)$), which encodes the size of the mapping returned by P as percentage of $|S \times T|$.

COMPUTING $\gamma(P)$ AND $s(P)$ FOR ATOMIC LS: Several approaches can be envisaged to achieve this goal. In our implementation of HP, we used approximations based on sampling. The basic assumption behind our feature choice was that LD frameworks are usually agnostic of S and T before the beginning of the LD. Thus, we opted for approximating the runtime of atomic plans P by using $|S|$ and $|T|$ as parameters. We chose these values because they be computed in linear time.[5] To approximate $\gamma(P)$ for atomic plans, we generated source and target datasets of sizes $1000, 2000, \ldots, 10000$ by sampling data from the English labels of DBpedia 3.8. We then stored the runtime of the measures implemented by our framework for different thresholds θ between 0.5 and 1.[6] The runtime of the i^{th} experiment was stored in the row y_i of a column vector Y. The corresponding experimental parameters $(1, |S|, |T|, \theta)$ were stored in the row r_i of a four-column matrix R. Note that the first entry of all r_i is 1 to ensure that we can learn possible constant factors. We finally computed the vector $\Gamma = (\gamma_0, \gamma_1, \gamma_2, \gamma_3)^T$ such that

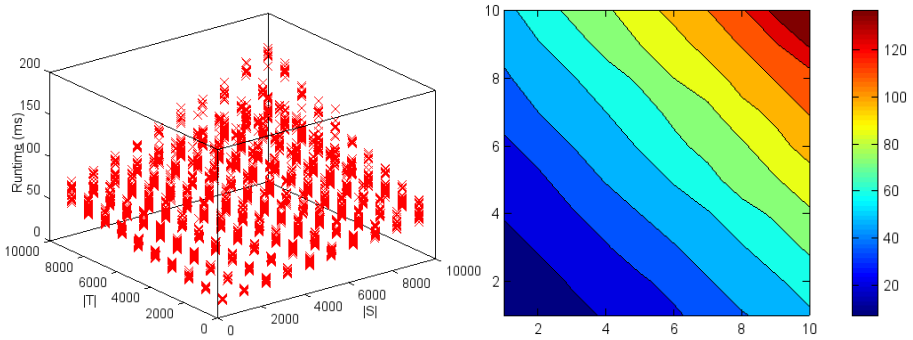
$$\gamma(P) = \gamma_0 + \gamma_1|S| + \gamma_2|T| + \gamma_3\theta. \tag{8.3}$$

To achieve this goal, we used the following exact solution to linear regression: $\Gamma = (R^T R)^{-1} R^T Y$. The computation of $s(P)$ was carried out similarly with the sole difference that the entries y_i for the computation of $s(P)$ were $\frac{|M_i|}{|S| \times |T|}$, where M_i is the size of the mapping returned by the i^{th} experiment. Figure 8.6 shows a sample of the results achieved by different algorithms in our experiments. The plan returned for the atomic LS$f(m, \theta)$ is `(run,m,θ)`.

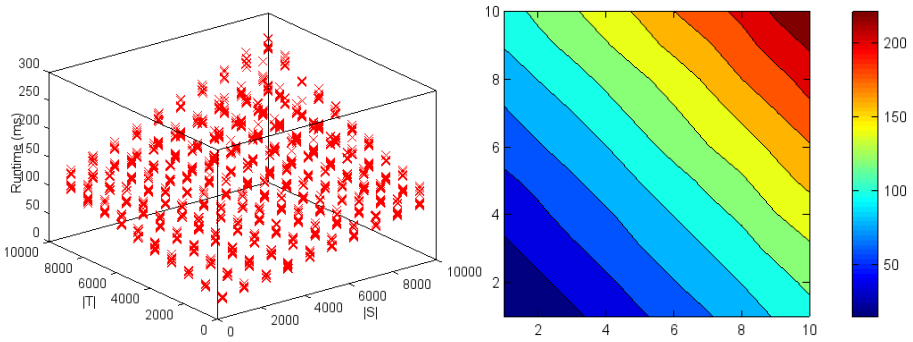
COMPUTING $\gamma(P)$ AND $s(P)$ FOR COMPLEX LS: The computation of the costs associated with atomic `filter`, `filterout` and operators was computed analogously to the computation of runtimes for atomic LS. For filters, the feature was the size of the input mapping. For non-atomic plans P, we computed $\gamma(P)$ by summing up the $\gamma(p_i)$ for all the steps p_i included in the plan. The selectivity of operators was computed based on the selectivity of the mappings that served as input for the operators. To achieve this goal, we assumed that the selectivity of a plan P to be the probability that a pair (s, t) is returned after the execution of P. Moreover, we assumed the input mappings M_1 (selectivity: s_1) resp. M_2 (selectivity: s_2) to be the results of independent computations. Based on these assumptions, we derived the following selectivities for $op(M_1, M_2)$:

5 Other values can be used for this purpose but our results suggest that using $|S|$ and $|T|$ is sufficient in most cases.

6 We used the same hardware as during the evaluation.



(a) Runtimes for `trigrams`. (b) Heatmap for `trigrams`.



(c) Runtimes for `levenshtein`. (d) Heatmap for `levenshtein`.

Figure 8.6: Runtimes achieved by PPJoin+ (`trigrams`) and EDJoin (`levenshtein`) for $\theta = 0.5$. The x-axis of the heatmap show |S| in thousands, while the y-axis shows |T| in thousands. The color bars show the runtime in ms.

- $op = \cap \rightarrow s(op) = s_1 s_2$.
- $op = \cup \rightarrow s(op) = 1 - (1 - s_1)(1 - s_2)$.
- $op = \setminus \rightarrow s(op) = s_1(1 - s_2)$.

8.3.2.2 *The HP algorithm*

The core of the approach implemented by HP is shown in Algorithm 8.1. For atomic specifications $f(m, \theta)$, HP simply returns `(run,m,θ)` (GETBESTPLAN method in Algorithm 8.1). If different algorithms which allow running m efficiently are available, HP chooses the implementation that leads to the smallest runtime $\gamma(P)$. Note that the selectivity of all algorithms that allow running m is exactly the same given that they must return the same mapping. If the specification $L = (m, \theta, op(L_1, L_2))$ is not atomic, HP's core approach is akin to a divide-and-conquer approach. It first devises a plan for L_1 and L_2 and then computes the costs of different possible plans for op. For $\cap$ for example, the following three plans are equivalent:

1. **Canonical plan.** This plan simply consists of merging (via the CONCATENATE method in Algorithm 8.1) the results of the best plans for L_1 and L_2. Consequently, the plan consists of (1) running the best plan for L_1 (i.e., Q_1 in Algorithm 8.1), (2) running the best plan for L_2(i.e., Q_2 in Algorithm 8.1),

then (3) running the `intersection` action over the results of Q_1 and Q_2 and finally (4) running `filter` over the result of the intersection action.

2. **Filter-right plan.** This plan uses $f(m_2, \theta_2)$ as a filter over the results of Q_1. Consequently, the plan consists of (1) running the best plan for L_1, then (2) running the `filter` action with measure m_2 and threshold θ_2 over the results of Q_1 and finally (3) running `filter` with measure m and threshold θ over the previous result.

3. **Filter-left plan.** Analogous to the filter-right plan with L_1 and L_2 reversed.

Similar approaches can be derived for the operators $\cup$ and $\setminus$ as shown in Algorithm 8.1. HP now returns the least costly plan as result (GETLEASTCOSTLY method in Algorithm 8.1). This plan is finally forwarded to the execution engine which runs the plan and returns the resulting mapping.

Given that the alternative plans generated by HP are equivalent and that HP always chooses one of this plan, our algorithm is guaranteed to be complete. Moreover, HP approximates the runtime of at most 3 different plans per operator and at most k different plans for each leaf of the input specification (where k is the maximal number of algorithms that implements a measure m in our framework). Consequently, the runtime complexity of HP is $O(\max\{k, 3\} \times |L|)$.

8.4 EVALUATION

8.4.1 *Experimental Setup*

The aim of our evaluation was to measure the runtime improvement of HELIOS the overall runtime of LS. We thus compared the runtimes of EP_0 (i.e., LIMES), RW (i.e., RW + EP_0), HP and HELIOS (i.e., RW +HP) in our experiments. We chose LIMES because it has been shown to be very time-efficient in previous work (Ngonga Ngomo, 2012b). We considered manually created and automatically generated LS. All experiments were carried out on server running Ubuntu 12.04. In each experiment, we used a single kernel of a 2.0GHz AMD Opteron processor with 10GB RAM.

The manually created LS were selected from the LATC repository.[7] We selected 17 LS which relied on SPARQL endpoints that were alive or on data dumps that were available during the course of the experiments. The specifications linked 18 different datasets and had sizes between 1 and 3. The small sizes were due to humans tending to generate small and non-redundant specifications.

The automatic specifications were generated during a single run of specification learning algorithm EAGLE (Ngonga Ngomo and Lyko, 2012) on four different benchmark datasets described in Table 8.3.[8] The mutation and crossover rates were set to 0.6 while the number of inquiries per iteration was set to 10. The population size was set to 10. The sizes of the specifications generated by EAGLE varied between 1 and 11. We compared 1000 LS on the OAEI 2010 Restaurant and the DBLP-ACM dataset each, 80 specifications on the DBLP-Scholar dataset and 100 specifications on LGD-LGD. We chose to use benchmark datasets to ensure

7 https://github.com/LATC/24-7-platform/tree/master/link-specifications

8 The Restaurant data is available at http://oaei.ontologymatching.org/2010/. DBLP-ACM and DBLP-Scholar are at http://dbs.uni-leipzig.de/en/research/projects/object_matching/fever/benchmark_datasets_for_entity_resolution.

Algorithm 8.1 The PLAN method

```
if L is atomic then
  P = getBestPlan(L);
else
  if L = f(m, θ, op(L1)) then
    P := getBestPlan(L1)
  else
    Q1 := plan(L1)
    Q2 := plan(L2)
    if L = f(m, θ, ∩(L1, L2)) then
      P0 := concatenate(intersection, Q1, Q2)
      P1 := concatenate(filter(m1, θ1), Q2)
      P2 := concatenate(filter(m2, θ2), Q1)
      P := getLeastCostly(P0, P1, P2)
    else if L = f(m, θ, ∪(L1, L2)) then
      P0 := concatenate(union, Q1, Q2)
      P1 := concatenate(union, filter(m2, θ2, S × T) , Q2)
      P2 := concatenate(union, filter(m1, θ1, S × T) , Q1)
      P := getLeastCostly(P0, P1, P2)
    else if L = f(m, θ, \(L1, L2)) then
      P0 := concatenate(minus, Q1, Q2)
      P1 := concatenate(filterout(m2, θ2), Q2)
      P := getLeastCostly(P0, P1)
    end if
  end if
  a0 = filter(m, θ)
  P = concatenate(a0, P)
end if
return P
```

that the specifications used in the experiments were of high-quality w.r.t. the F-measure they led to. Each specification was executed 10 times. No caching was allowed. We report the smallest runtimes over all runs for all configurations to account for possible hardware and I/O influences.[9]

8.4.2 *Results on Manual Specifications*

The results of our experiments on manual specifications are shown in Table 8.2 and allow deriving two main insights: First, HELIOS can improve the runtime of atomic specifications (which made up 62.5% of the manual LS). This result is of tremendous importance as it suggests that the overhead generated by HELIOS is mostly insignificant, even for specifications which lead to small runtimes (e.g., DBP-DataGov requires 8ms). Moreover, our experiments reveal that HELIOS achieves a significant improvement of the overall runtime of specifications with sizes larger than 1 (37.5% of the manual LS). In the best case, HELIOS is 49.5 times faster than EP_0 and can reduce the runtime of the LS LDG-DBP (A) from 52.7s to 1.1s by using a filter-left plan. Here, we see that the gain in runtime generated by

9 All evaluation results can be found at https://github.com/AKSW/LIMES.

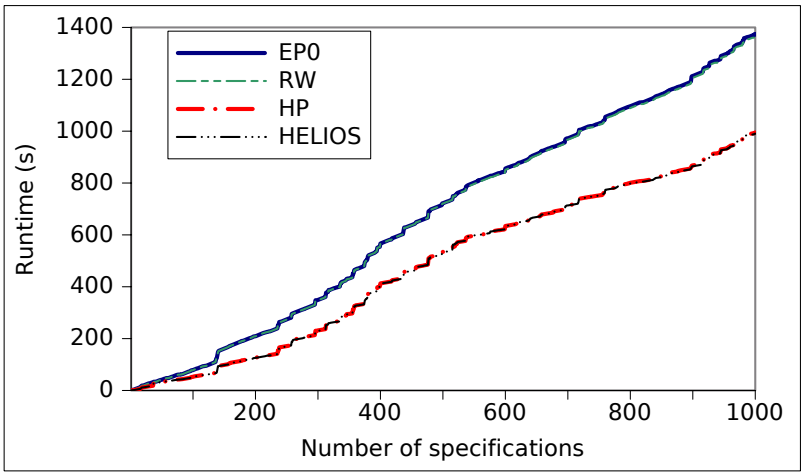
HELIOS grows with $|S| \times |T|$. This was to be expected as a good plan has more effect when large datasets are to be processed. Overall, HELIOS outperforms LIMES' canonical planner on all non-atomic specifications. On average, HELIOS is 4.3 times faster than the canonical planner on LS of size 3.

Source - Target	$\|S\| \times \|T\|$	EP_0 (ms)	RW (ms)	HP (ms)	HELIOS (ms)	Gain (ms)
DBP - Datagov	1.7×10^3	**8**	**8**	**8**	**8**	0
RKB - DBP	2.2×10^3	**1**	**1**	**1**	**1**	0
Epo - DBP	73.0×10^3	54	**53**	54	**53**	1
Rail - DBP	133.2×10^3	269	**268**	**268**	**268**	1
Stad - Rmon	341.9×10^3	25	23	15	**14**	11
EVT - DF (E)	531.0×10^3	**893**	906	909	905	-12
Climb - Rail	1.9×10^6	41	**40**	**40**	**40**	1
DBLP - DataSW	92.2×10^6	59	59	58	**54**	5
EVT - DF (P)	148.4×10^6	2,477	2,482	2,503	**2,434**	43
EVT - DBLP	161.0×10^6	9,654	**9,575**	9,613	9,612	42
DBP - OpenEI	10.9×10^3	**2**	**2**	**2**	**2**	0
DBP - GSpecies	94.2×10^3	120	**119**	120	**119**	1
Climb - DBP	312.4×10^3	**55**	**55**	**55**	**55**	0
DBP - LGD (E)	34.1×10^6	2,259	2,133	**1,206**	1,209	1,050
Climb - LGD	215.0×10^6	24,249	24,835	**3,497**	3,521	20,728
DBP - LGD (A)	383.8×10^6	52,663	59,635	1,066	**1,064**	51,599
LGD - LGD	509.3×10^9	46,604	38,560	32,831	**22,497**	24,107

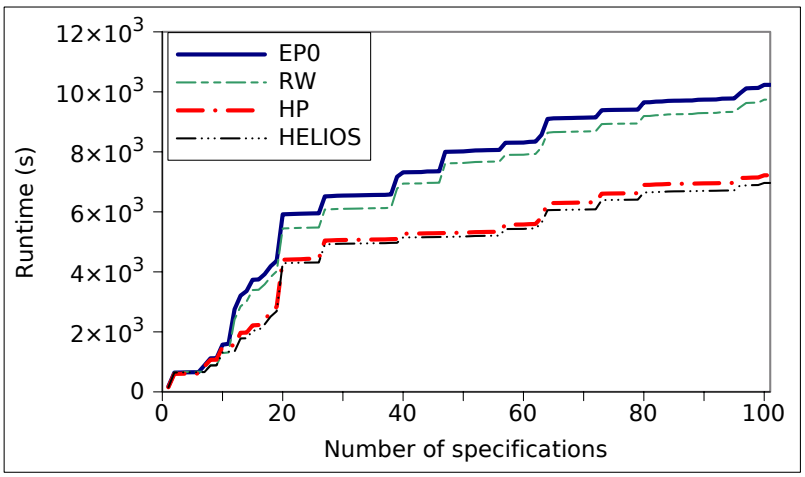
Table 8.2: Comparison of runtimes on manual specifications. The top portion of the table shows runtimes of specifications of size 1 while the bottom part shows runtimes on specifications of size 3. EVT stands for Eventseer, DF for DogFood, (P) stands for person, (A) stands for airports, (U) stands for universities, (E) stands for events. The best runtimes are in bold.

8.4.3 *Results on Automatic Specifications*

Overall, our results on automatic specifications show clearly that HELIOS outperforms the state of the art significantly. In Table 8.3, we show the average runtime of EP_0, RW, HP and HELIOS on four different datasets of growing sizes. The overall runtime of HELIOS is clearly superior to that of EP_0 on all datasets. As expected, the gain obtained by using HELIOS grows with the size of $|S| \times |T|$. In particular, the results on the very small Restaurant dataset support the results achieved on the manual specifications. While HP alone does not lead to a significant improvement, HELIOS leads to an improvement of the overall runtime by 6.35%. This improvement is mostly due to RW eliminating filters and therewith altering the plans generated by HP. These alterations allow for shorter and more time-efficient plans.



(a) DBLP-ACM



(b) LGD-LGD

Figure 8.7: Cumulative runtimes on DBLP-ACM and LGD-LGD

On the larger DBLP-ACM dataset, HELIOS achieve a runtime that is up to 185.8 times smaller than that of EP_0 (e.g., for f(∩(f(`jaccard`(authors, authors), 0.93), f(`edit`(venue, venue), 0.93)),0.53)). Yet, given that the runtime approximations are generic, HELIOS sometimes generated plans that led to poorer runtimes. In the worst case, a plan generated by HELIOS was 6.5 times slower than the plan generated by EP_0 (e.g., for f(∩(f(`edit`(authors, authors), 0.59), f(`cosine`(venue,venue),0.73)),0.4)). On average, HELIOS is 38.82% faster than EP_0. Similar results can be derived from DBLP-Scholar, where HELIOS is 29.61% faster despite having run on only 80 specifications. On our largest dataset, the time gain is even larger with HELIOS being 46.94% faster. Note that this improvement is very relevant for end users, as it translates to approximately 1h of runtime gain for each iteration of our experiments. Here, the best plan generated by HELIOS is 314.02 times faster than EP_0. Moreover, we can clearly see the effect of RW with average runtime improvement of 5.1% (see Figure 8.7).

We regard our overall results as very satisfactory given that the algorithms underlying EP_0 are in and of themselves already optimized towards runtime. Still, by combining them carefully, HELIOS can still cut down the overall runtime of learning algorithms and even of manually created link specifications. To ensure that

	$\|S\| \times \|T\|$	$\|L\|$	F_1	EP_0	RW	HP	HELIOS
Restaurants	72.3×10^3	4.44±1.79	0.89	0.15	0.15	0.15	**0.14**
DBLP-ACM	6.0×10^6	6.61±1.32	0.99	1.38	1.37	1.00	**0.99**
DBLP-Scholar	168.1×10^6	6.42±1.47	0.91	17.44	17.41	13.54	**13.46**
LGD-LGD	5.8×10^9	3.54±2.15	0.98	102.33	97.40	72.19	**69.64**

Table 8.3: Summary of the results on on automatically generated specifications. $|L|$ shows for the average size ± standard deviation of the specifications in the experiment. F_1 shows the F-measure achieved by EAGLE on the dataset. The runtimes in four rightmost columns are the average runtimes in seconds.

our improvements are not simply due to chance, we compared the distribution of the cumulative runtimes of EP_0 and RW, HP and HELIOS as well EP_0 and HELIOS by using a Wilcoxon paired signed rank test at a significance level of 95%. On all datasets, all tests return significant results, which shows that the RW, HP and HELIOS lead to statistically significant runtime improvements.

8.5 RELATED WORK

The task we address shares some similarities with the task of query optimization in databases (Silberschatz et al., 2006). A large spectrum of approaches have been devised to achieve this goal including System R's dynamic programming query optimization (Selinger et al., 1979), cost-based optimizers and heuristic optimizers (Kanne and Moerkotte, 2010) and approaches based on genetic programming (Bennett et al., 1991). HELIOS is most tightly related to heuristic optimizers as it relies on an optimality assumption to discover plans in polynomial time. Overviews of existing approaches can be found in (Chaudhuri, 1998; Silberschatz et al., 2006). The main difference between the task at hand and query optimization for databases are as follows: First, databases can store elaborate statistics on the data they contain and use these to optimize their execution plan. LD frameworks do not have such statistics available when presented with a novel LS as they usually have to access remote data sources. Thus, HELIOS must rely on statistics that can be computed efficiently while reading the data. Moreover, our approach also has to rely on generic approximations for the costs and selectivity of plans. Still, we reuse the concepts of selectivity, rewriting and planning as known from query optimization in databases.

This work is a contribution to the research area of LD. Several frameworks have been developed to achieve this goal. The LIMES framework (Ngonga Ngomo, 2012b), in which HELIOS is embedded, provides time-efficient algorithms for running specific atomic measures (e.g., PPJoin+ (Xiao et al., 2008) and $\mathcal{HR}^3$ (Ngonga Ngomo, 2012a)) and combines them by using set operators and filters. While LIMES relied on graph traversal until now, most other systems rely on blocking. For example, SILK (Isele et al., 2011) relies on MultiBlock to execute LS efficiently. Multiblock allows mapping a whole link specification in a space that can be segmented to overlapping blocks. The similarity computations are then carried out within the blocks only. A similar approach is followed by the KnoFuss system (Nikolov et al., 2012). Other time-efficient systems include (Song and Heflin,

2011) which present a lossy but time-efficient approach for the efficient processing of LS. Zhishi.links on the other hand relies on a pre-indexing of the resources to improve its runtime (Niu et al., 2011). CODI uses a sampling-based approach to compute anchor alignments to reduce the its runtime (Huber et al., 2011). Other systems descriptions can be found in the results of the Ontology Alignment Evaluation Initiative (Euzenat et al., 2011).[10] The idea of optimizing the runtime of schema matching has also been considered in literature (Shvaiko and Euzenat, 2013). For example, (Peukert et al., 2010) presents an approach based on rewriting. Still, to the best of our knowledge, HELIOS is the first optimizer for link discovery that combines rewriting and planning to improve runtimes.

8.6 CONCLUSION AND FUTURE WORK

We presented HELIOS, the (to the best of our knowledge) first execution optimizer for LS. We evaluated our approach in manually created and automatically generated LS. Our evaluation shows that HELIOS outperforms the canonical execution planner implemented in LIMES by up to two orders of magnitude. Our approach was intended to be generic. Thus, we used generic evaluation functions that allowed to detect plans that should generally work. Our results suggest that using more dataset-specific features should lead to even better runtimes and higher improvements. We thus regard HELIOS as the first step in a larger agenda towards creating a new generation of self-configuring and self-adapting LD frameworks. During the development of HELIOS, we noticed interesting differences in the behaviour of LD algorithms for different languages. For example, the Γ vector for the different measures differs noticeably for French, English and German. We will investigate the consequences of these differences in future work. Moreover, we will investigate more elaborate features for approximating the selectivity and runtime of different algorithms.

10 http://ontologymatching.org

9 PARALLEL IMPLEMENTATION OF LIMES

9.1 PREAMBLE

After investigating the runtime of link discovery approaches on single cores, we now delve into parallel implementations for link discovery. In this chapter, we study the implementation of the LIMES approach presented in Chapter 3 within the Map-Reduce paradigm. The content of the chapter is taken from (Hillner and Ngonga Ngomo, 2011) and was the result of a thesis supervised by the author, who also developed the idea, implemented a portion of the code behind the chapter and co-wrote the publication.

9.2 INTRODUCTION

The Linked Data Web has evolved from 12 knowledge bases in May 2007 to 203 knowledge bases in September 2010, i.e., in less than four years (Heath and Bizer, 2011). Currently, the Linked Open Data cloud consists of more than 25 billion triples, of which less than 3% are links between knowledge bases. Yet, links between knowledge bases play a central role in tasks such as cross-ontology question answering (Lopez et al., 2009) and large-scale inferences (Urbani et al., 2010). The mere size of the Linked Data Cloud makes manual linking impossible. For examples, manually checking the films in DBpedia (Auer et al., 2007) for duplicates would require approximately 2.4×10^9 comparisons to complete. Assuming that a human could compare 1 pair of entities per second, he would still need more than 70 years to carry out such a task. Link Discovery Frameworks have been developed over the last years to mitigate this problem. Yet, even runtime-optimized approaches such as SILK (Version 2.2) (Volz et al., 2009) or LIMES (Version 0.3.2) (Ngonga Ngomo and Auer, 2011) still require several billion comparisons and thus a considerable amount of time to complete such a task. Furthermore, large mapping tasks require significant amounts of memory and are thus prone to leading to buffer overflows when executed on standard hardware. Consequently, link discovery can be classified as a high performance computing (HPC) problem and needs to be handled as such.

Two main categories of remedies can be envisaged to mitigate the time complexity of Link Discovery: developing highly efficient algorithms and distributing the computation effort across several machines. (Ngonga Ngomo and Auer, 2011) presents such a time-efficient algorithm for Link Discovery. In this paper, we present a minimally invasive extension of this given algorithm (as implemented in the LIMES[1] framework) from a single-core version to the parallelized framework for Link Discovery dubbed LIMES M/R. We first study the requirements to an effective parallelization of the framework at hand. Thereafter, we present our approach to parallelizing LIMES and give some insights in the current implementation of LIMES M/R. We evaluate the runtime of our approach against the single-core version of LIMES and the parallelized version of SILK in ten settings.

1 http://limes.sf.net

We show that we achieve a speedup of up to 11 and that we therewith outperform the parallelized version of SILK significantly.

The remainder of this paper is structured as follows: In Section 9.3, we give a brief overview of related work. Parallelization paradigms and models out of which we chose the most suitable for implementing a parallel version of the LIMES approach are presented in Section 9.4. In the subsequent section, Section 9.5, we present the current implementation of LIMES M/R. Section 9.6 then presents 12 experiments carried out with LIMES M/R and gives insights in the performance of our approach. Finally, we conclude in Section 9.7.

9.3 RELATED WORK

Link Discovery is a central task within the Linked Data setting. The most common approach to Link Discovery is based on comparing certain property values of the entities from a source knowledge base S and a target knowledge base T that are to be linked. A link is then generated if and only if the distance between these properties is below a given threshold θ. Several approaches have already been developed for discovering links between datasets. (Ngonga Ngomo and Auer, 2011) points out that two main categories of Link Discovery frameworks can be differentiated: *domain-specific* and *universal* frameworks. Domain-specific LD frameworks aim at discovering links between knowledge bases from a particular domain such as universities and conferences (RKB-CRS, (Glaser et al., 2009)) as well as music (GNAT, (Raimond et al., 2008)).

Of higher relevance for this paper are universal LD frameworks, which are designed to carry out mapping tasks independently from the domain of the source and target knowledge bases and can thus be confronted with very large link discovery tasks. RDF-AI (Scharffe et al., 2009) implements a five-step approach that comprises the preprocessing, matching, fusion, interlinking and post-processing of datasets. SILK (Volz et al., 2009) (Version 2.2) implements blocking and rough index pre-matching to reach a quasi-linear time-complexity. In addition, the parallelized version of SILK runs on Hadoop, which significantly improves the total runtime of the system. The LIMES approach (Ngonga Ngomo and Auer, 2011) presupposes that the datasets to link are in a metric space. It then uses the triangle inequality to partition the metric space. Each portion of the metric space is represented by a so-called exemplar, which is used as reference point to compute a pessimistic approximation of distances within that given portion of space. Based on these approximations, LIMES can discard a large number of similarity computations without loosing links. Although the LIMES framework displays significant runtime improvements, it still requires a considerable amount of time to carry out mappings of the order of 10^9 comparisons. In this paper, we show how the time-optimized approach implemented in LIMES can be rendered even faster via parallelization. We also evaluate the total runtime of the parallelized version of LIMES against that of SILK and show that we outperform it in all settings.

9.4 PARALLELIZATION PARADIGMS

Building programs for parallel computation has been a challenge since parallel computation was invented. Yet during the last decade, the rapid technological development on the hardware side has led to parallel programming becoming in-

creasingly important in order to use the full potential, even of standard computers (Asanovic et al., 2006; Chen and Bairagi, 2010). In (Kasim et al., 2008), two main approaches for application parallelization are identified, namely *auto parallelization* an *parallel programming*.

The *auto-parallelization approach*, as realized mainly by parallel compilers, can be applied to automatically parallelize applications that were implemented to run linearly. Yet, previous works show that automatically parallelized applications are limited in their degree of parallelism and thus led to a limited speedup of the application (Hennessy and Patterson, 2007).

To ensure a higher parallelism, parallel programming techniques need to be applied. There are various parallel programming models from which an application developer can choose (Asanovic et al., 2006; Chen and Bairagi, 2010; Ebnenasir and Beik, 2009). (Kasim et al., 2008) identified seven evaluation criteria for parallel programming models and evaluated six models against these criteria. In (Asanovic et al., 2006), five critical tasks during the development of parallel applications are named and ten programming models are evaluated against them. Out of these two sets of criteria, the following five are of central importance when parallelizing link discovery frameworks:

Task-to-Worker Mapping defines which task shall be executed on which worker. One can distinguish automatic (done by the runtime-system) and manual (to be managed by the developer) worker mapping.

Worker Management is the management of the workers, e. g., their life cycles. Automatic management indicates that the system controls the creation or termination of, for example, threads or processors. In manual management, the developer has to maintain the worker life cycles manually.

Data Distribution describes the task of distributing data over the parallel system to feed the workers with input data to be processed and collect the results from the workers. Again, one can distinguish automatic data distribution and manual data distribution where the developer has to manage the passing of data or the current state of the data.

Synchronization is the task of managing the order of the workers which access shared data. Automatic synchronization indicates that the developer does not need to worry about synchronization details.

Programming Methodologies defines how the parallelism is abstracted, as an example, the available API and the language specification. This can be a main factor in development time when a developer, for instance, uses a new programming model that does not offer an intuitive language.

In the following, we give a brief overview of four current parallel programming models and select the most accurate for the task at hand by using the criteria defined above.

9.4.1 *Parallel programming models*

9.4.1.1 *OpenMP*

Open Multi-Processing (OpenMP)[2] (Quinn, 2003) is an open specification which extends C, C++ and Fortran with parallelization capabilities by offering a set of compiler directives and callable runtime libraries. In OpenMP, the workers are realized as threads whose management is implicit. This means that the developer does not need to manage the life cycle of the threads. Instead, OpenMP provides special compiler directives which indicate that a code section, e. g., a for-loop, is to be executed in parallel. The task-to-worker mapping is managed by OpenMP. Since OpenMP is a programming model for shared-memory architectures, the effort for distributing data between the workers is small. Each worker can access the data in the shared address space. Yet, shared memory renders the synchronization of the access of the shared data difficult. In OpenMP, the synchronization is done implicitly by several mechanisms and the programmer only has to declare sections where a synchronization is needed. OpenMP abstracts away the most critical tasks from the programmer, like the the workload partitioning or the synchronization mechanisms.

9.4.1.2 *MPI*

The Message Passing Interface (MPI)[3] (Quinn, 2003) is a standard for inter-process communication in distributed computer systems. Developers can use MPI for the communication in distributed memory environments as well as shared memory computers. MPI has been implemented for languages such as C, C++, Fortran and Java.[4] In contrast to OpenMP, workers in MPI are instantiated as processes. Still, the worker management is mostly taken care of by the MPI runtime. All the application needs to specify is the number processes needed for the current task. The task-to-worker mapping, workload partitioning and data distribution have to be implemented explicitly. The synchronization in MPI is carried out by the MPI engine but the developer still has to define where synchronized execution is needed.

9.4.1.3 *UPC*

The Unified Parallel C (UPC)[5] (El-Ghazawi and Cantonnet, 2002) programming language is an extension of the C programming language and is intended to be used for parallel programming on HPC machines. It supports both shared memory machines and distributed memory environments. In UPC, the workers are threads and the worker management is done implicitly. All the application needs to specify is the number of threads that are necessitated. The workload partitioning and task-to-worker mapping can be carried out implicitly or explicitly. The synchronization in UPC is more complex than in OpenMP or MPI and offers implicit and explicit synchronization constructs. UPC abstracts from the memory architecture and presents the memory as a partitioned global address space. Furthermore, the UPC-API and the language specification provides various mechanisms for implicit

2 http://openmp.org/wp/
3 http://www.mcs.anl.gov/research/projects/mpi/
4 http://www.hpjava.org/mpiJava.html
5 http://upc.lbl.gov/

and explicit worker mapping or workload partitioning. This flexibility bears the obvious advantage that the developer can choose to handle single problems manually, e. g., the memory access synchronization, but does not have the need to.

9.4.1.4 *MapReduce*

MapReduce is another programming model for HPCs. The core idea of this programming model is based on the *Map* and *Reduce* primitives of functional programming languages. The workers in MapReduce are implemented as processes and managed by the MapReduce cluster. The developer does not have to care about the creation or destruction of workers but there are various options that can be used to configure the cluster, such as the number of map, combine, or reduce tasks. Furthermore, there is no need to specify a task-to-worker mapping during development time. MapReduce maps each task to an available worker and re-executes tasks on other workers if necessary. This ensures a very high fault tolerance and can speed-up the computation in case of slow workers being part of the cluster. The distribution of the input and the generated intermediate data can be carried out fully automatically, but there are also possibilities to influence the distribution over the cluster. MapReduce hides nearly every task of parallel programming by providing Interfaces against which the programmer can develop his application.

9.4.2 *Comparison*

Table 9.1 shows a comparison of the parallel programming models made above. Each criterion is rated as *automatic* if this task is fulfilled by the system, *manual* if the developer has to handle this task manually or *both* if the task is handled by the system but the programmer can choose to take over the task. An exception to the evaluation is the criterion of the programming methodologies where only positive and negative scorings are possible. There for instance, a scoring of "++" means that the programming model hides most of the parallelization tasks which enables the developer to get a quick start with the programming model and sets the main focus to the development of the core-features of the application.

Table 9.1 shows that the programming models OpenMP and MapReduce are best suited for parallelizing LIMES. Given that manual data synchronization is not critical, as the data for linking is partitioned and is consequently easy to synchronize, we chose to use MapReduce as it requires a minimal amount of effort for the cluster setup, is supported by a large number of service providers and can be executed on multi-core machines as well as HPCs without any adaptation.

	Worker management	**Task-to-worker mapping**	**Data distribution**	**Sync.**	**Programming methodologies**
OpenMP	automatic	automatic	automatic	automatic	++
MPI	automatic	manual	manual	automatic	+/-
UPC	automatic	both	automatic	both	+
MapReduce	automatic	automatic	automatic	manual	++

Table 9.1: Evaluation of parallel programming models. Sync. stands for synchronsation

9.5 LIMES M/R

In its stand-alone and not parallel version, LIMES implements the Link Discovery Process depicted in Figure 9.1 and explicated in (Ngonga Ngomo and Auer, 2011). It is obvious that several of these steps can be executed in parallel, for example, fetching the source and target data and filling the corresponding caches. Yet, these steps are usually not the bottleneck of Link Discovery Frameworks. Given a large-scale link discovery task, most of the runtime of any link discovery framework is spent computing the similarity between instances from the source and target knowledge base. Thus, we focus in this paper on parallelizing the similarity computation step. The resulting process is shown in Figure 9.2. Overall, the process implemented by LIMES M/R can be subdivided into two main steps: initialization and execution.

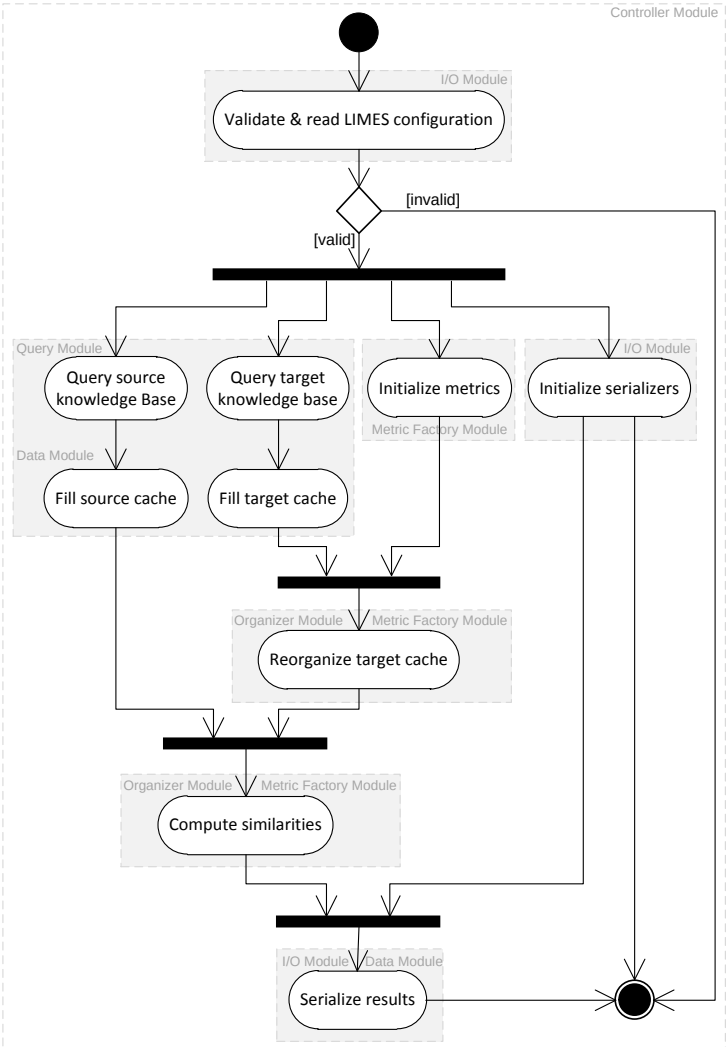


Figure 9.1: Activity diagram of LIMES

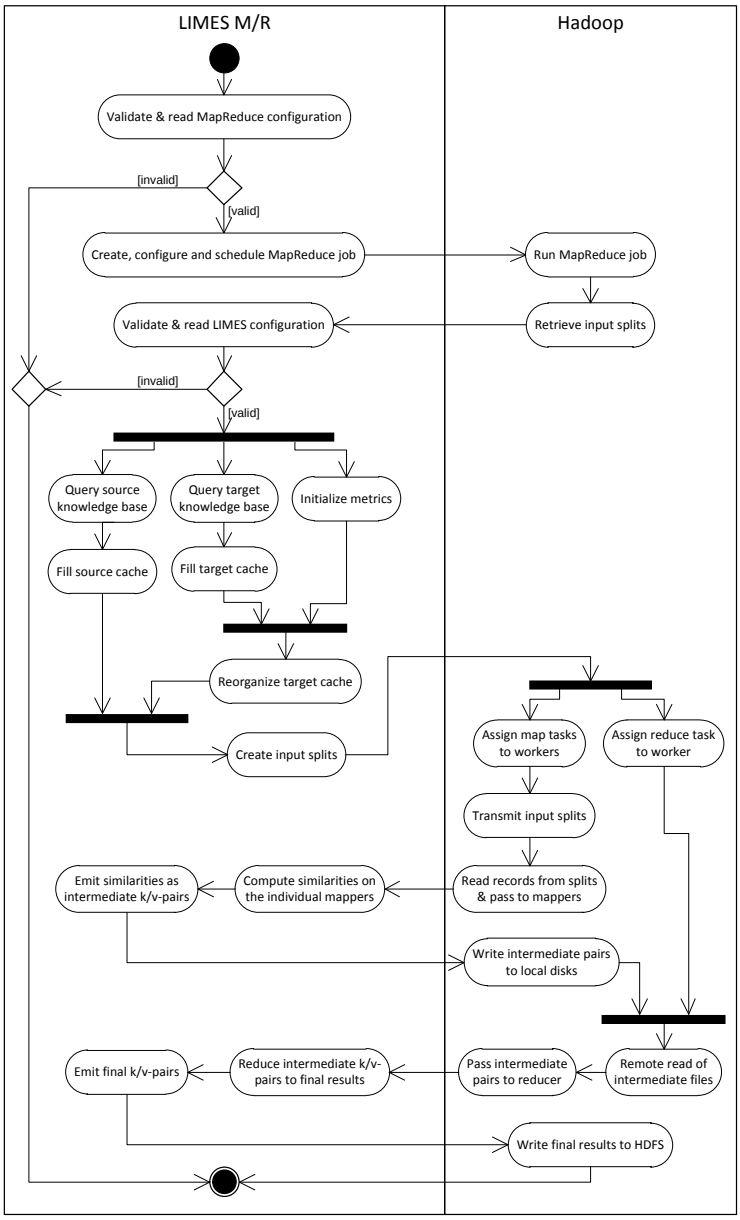


Figure 9.2: Activity diagram of LIMES M/R

9.5.1 *Initialization*

LIMES M/R first checks the MapReduce configuration file that is passed in addition to the LIMES configuration file. If the configuration is valid, the next step of the process creates and configures the MapReduce job by using the information in the configuration file before scheduling the job for execution at the MapReduce framework. Should the configuration file be invalid, then the link discovery process is aborted.

Once a job is scheduled for execution, Hadoop starts the job by executing the MapReduce workflow as depicted in Figure 9.3. During this job execution the execution of atomic activities switches between LIMES M/R and Hadoop several times.

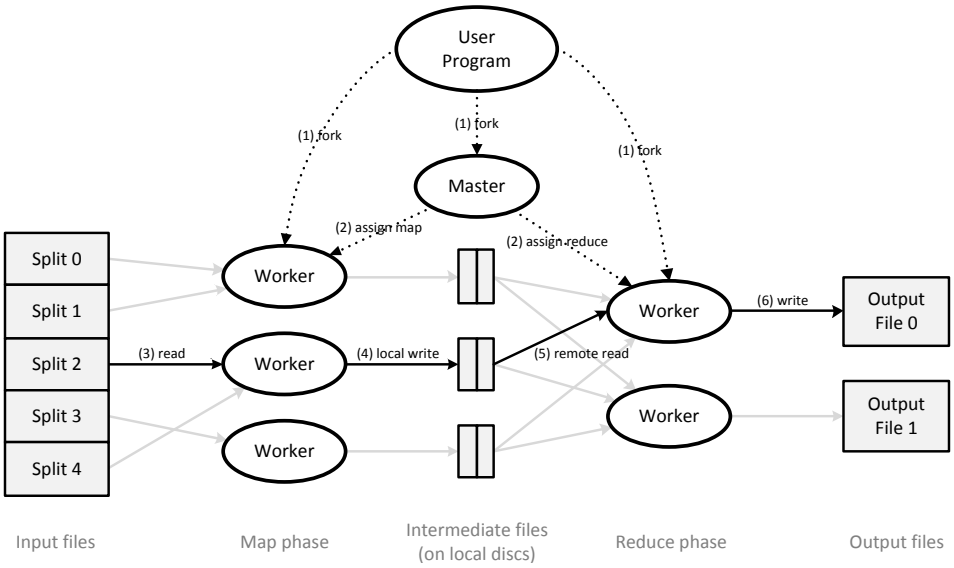


Figure 9.3: Workflow of a MapReduce program (Dean and Ghemawat, 2004, 2008)

The MapReduce job takes care of the distribution of the program to the master and worker nodes. The job execution per se begins with the retrieval of the input splits that are later passed to the mappers of the job in order to find links between the input knowledge bases. This activity is executed on the master node of the cluster and is carried out as follows: First, the LIMES configuration file is validated against its DTD and read. If the validation fails, the framework stops the execution of the job and returns with an appropriate error message. In the other case, the source and target knowledge bases are queried and instances matching the restrictions are extracted to the source and target caches. Furthermore, the underlying metrics to be used for the instance comparisons are initialized.

9.5.2 *Execution*

9.5.2.1 *Computation of Exemplars*

As soon as the target cache is filled and the metrics are initialized, LIMES M/R starts with the reorganization of the target cache as described in (Ngonga Ngomo and Auer, 2011). After the completion of the reorganization of the target cache, the computation of the input splits is started. The number and thus size of the input splits is thereby mainly influenced by the number of mappers of the job. The current implementation of LIMES M/R splits the source cache into a number of cache fragments which equals the number of mappers defined by the user. These source cache fragments are then used as the input keys for the map tasks of the job. We implemented two different variations of input split generators. The first implementation (LIMES M/R) uses the whole reorganized target cache as the value for each input key/value pair, while the second (LIMES M/R V2) splits the reorganized target cache into 10 fragments and uses these fragments as values for the input splits. In the latter case, the current implementation creates 10 times more map tasks than specified by the user.

9.5.2.2 *Split processing*

The input splits are processed within the MapReduce framework as depicted in the Hadoop Swimlane of Figure 9.2. The MapReduce framework then goes through a series of procedures. First, the map and reduce tasks are assigned to the worker nodes of the cluster. The implemented standard setting for the number of reducers that are used to join the intermediate key/value pairs coming from the mappers uses one reducer in order to join the mapper output into a single file lying in the distributed file system. In contrast, a MapReduce cluster usually runs a large number of map tasks with the aim at distributing the work over the whole cluster and ensuring a good work load balance. As stated in the previous section, the number of map tasks can be influenced by the user. However, Hadoop uses the generated input splits to feed the assigned map tasks with input data, but in contrast to most other MapReduce programs, LIMES M/R does not store its queried input data or the generated input splits in the distributed file system and passes their locations to the mappers. LIMES M/R transfers the generated input splits directly to the map tasks in order to build in-memory caches for the mappers. This is even for huge knowledge bases possible since the problem size is reduced significantly for the single map tasks by means of the input splitting. Due to the direct transferring of the input splits, the activity diagram contains an activity named *"Transmit input splits"* instead of a *"Read input splits"* activity. Next, Hadoop calls the `RecordReader` for each map task and input split that shall extract the single key/value pairs from the splits in order to be consumed by the mappers. The current implementation does not apply any optimizations to this step and simply creates the caches out of the passed binary input data coming from the input splits.

The next two steps are executed under the responsibility of LIMES M/R and are therefore depicted in the LIMES M/R Swimlane. The similarity computation process follows the one that has been introduced by LIMES and is executed in each map task for the input split that has been passed to it. In short, each instance of the key-cache (a fragment of the original source cache) is at first compared to each exemplar of the reorganized target cache or its fragment, depending on the version of LIMES M/R, as described for the input split generation. If the similarity lies above a given threshold and the triangle inequality-property of the metric space can be held, the instance is compared to each instance in the subspace of the exemplar, as long as the inequality can be applied. As soon as the similarity computation for one instance of the source cache fragment is completed, the mapper emits the resulting key/value pairs as intermediate pairs to the Hadoop output collector which then writes the results to the local disks of the worker that is executing the current map task.

9.5.2.3 *Reduction*

In the following, the reduce task is applied as soon as one map task produces intermediate key/value pairs. The job tracker now sends the locations of the intermediate results in the distributed file system to the worker node that runs the reduce task of LIMES M/R which then reads these pairs from the file system using Remote Method Invocation (RMI). This is all done internally by Hadoop and is therefore depicted in the Hadoop swimlane of the activity diagram. Once, the intermediate files are fetched, the reduce worker passes the intermediate key/value pairs to the reduce task where these pairs are reduced to the final output of the

framework. The reducer of LIMES M/R has to execute just one simple task, the combination of the incoming intermediate results to just one final result file. For this reason, the reducer simply passes all incoming pairs as plain text to the output collector of Hadoop which writes them into a single file on the distributed file system. If no further error occurs during the link discovery process, the execution of the LIMES M/R job successfully terminates after the last map task has finished its similarity computation and the so-produced intermediate results are passed to the reducer, which merges all results to one final set of links.

9.6 EXPERIMENTS AND RESULTS

To ensure the correctness and the effectiveness of the approach presented in the previous section, we carried ten experiments of different size.

9.6.1 *Experimental Setup*

The test cluster used in all experiments reported below was created by using the Amazon Elastic Compute Cloud (EC2) Web Service[6] and consisted of 9 computers[7] (nodes). One of these computers was exclusively used as the master node of the cluster and the other 8 nodes were designated as worker nodes of the cluster (the slaves). Since the master node was not intended to perform large computations, this node was launched as a standard small EC2 instance (*m1.small*), while the slave nodes were launched as high-CPU medium EC2 instances (*c1.medium*) in order to provide enough computation resources.

The small instance was assigned 1.7GB of memory and 1 EC2 Compute Unit[8] which is provided by 1 virtual core with 1 EC2 Compute Unit. The high-CPU medium instances with 1.7GB of memory and up to 5 EC2 Compute Units which were provided by 2 virtual cores with 2.5 EC2 Compute Units each. Both instance types are built on a 32-bit platform architecture and provided a moderate I/O performance. The cluster nodes were launched with the 32-bit version of the Ubuntu 10.04 LTS (Lucid Lynx) Server Edition operating system. The MapReduce cluster was set up with Apache Hadoop in version 0.20.2 of January 2010. The cluster was configured to run at most 2 map tasks per node in parallel which results in a total number of 16 parallel map tasks for the whole cluster. This value has been chosen due to the hardware restrictions of the cluster nodes. To avoid memory problems carrying out larger link discovery tasks, the JVMs of the slave nodes were started with a maximum of 1024MB of memory.

We carried out two series of experiments. In the first series experiments, we performed four link discovery tasks using different settings for linkage. The setup for this first series of experiments is summarized in Table 9.2. In the second series of experiments, we were concerned with measuring the effect of a change of threshold on our approach. Thus, we used ith the same datasets and carried out a link discovery task with increasing thresholds.

6 http://aws.amazon.com/ec2/

7 Amazon EC2 provides compute capacity on demand. Using this cloud-based service, it is possible to create virtual computer instances matching the personal requirements.

8 Amazon has developed several measures for the CPU capacity of the instances. The term *EC2 Compute Unit* has been introduced in order to make the EC2 instance types comparable. An EC2 Compute Unit provides the equivalent CPU capacity of a $1.0 - 1.2$GHz AMD Opteron or Intel Xeon Processor of 2007.

	Diseases	Cities	Films
Source property	`dc:title`	`rdfs:label`	`rdfs:label`
Target property	`linkedct:condition_name`	`rdfs:label`	`rdfs:label`
|S|	23600	12600	49000
|T|	5000	12600	49000
Threshold	0.75	0.95	0.95

Table 9.2: Summary of experimental setup

	Diseases	Cities	Films
LIMES M/R	218.6	133.1	1204.5
LIMES M/R - V2	240.7	193.1	1257.5
SILK	336.0	346.5	5607.1

Table 9.3: Comparison of LIMES and SILK on 16 cores

9.6.2 *Results*

The results of our first series of experiments are shown in Figure 9.4. Our results clearly show that we outperform both LIMES (single-core) and SILK (parallelized version running on 16 cores) with respect to runtime and reach a speedup of up to 11 on the Diseases dataset (see Table 9.3). Interestingly, LIMES M/R V2 leads to longer runtimes than LIMES M/R with 16 workers. Overall, we obtain the best speedup with 16 workers as awaited. Setting the numbers of workers to a value above 16 leads to a higher communication overhead between the master and the slaves as well a higher effect of the network latency. The second series of experiments reveals that lowering the threshold leads to improved speedup values. This is simply due to the network overhead being constant for all experiments and the runtime being larger. Consequently, the relative runtime decreases, leading to better speedup values.

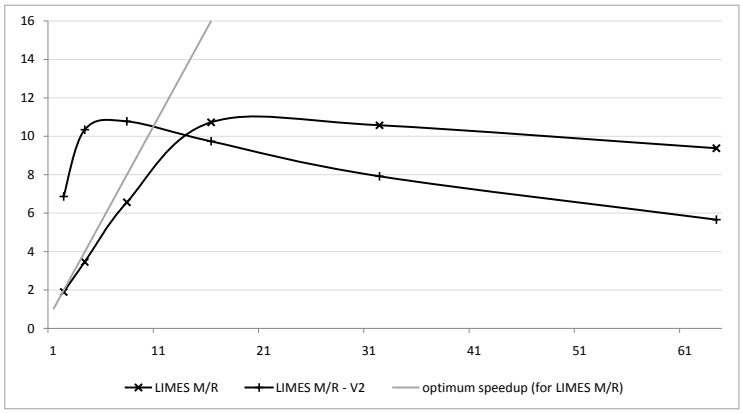
The speedup of LIMES could be improved by implementing a data portioning approach that makes better use of the distribution of data across the metric space. Currently, the exemplar computation is carried out without taking the skew within the data into consideration. Thus, it can lead to an unbalanced data distribution across the exemplars and thus to a sub-optimal distribution of the workload amongst the workers. An distribution-aware approach for exemplar computation still needs to be integrated in LIMES. Such an approach could then be used directly by LIMES M/R thank to the minimally invasive parallelization followed within this paper.

9.7 CONCLUSION AND FUTURE WORK

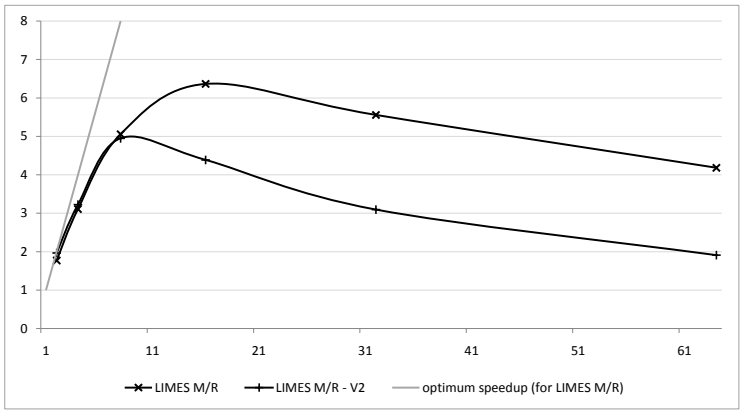
In this paper, we presented our extension of the LIMES framework using the Hadoop framework. We gave insights in the design principles and in the current implementation of LIMES M/R. We carry out ten experiments within which we compared LIMES, LIMES M/R and SILK. We showed that our implementation

leads to speedup values up to 11 on 16 processors. In all experiments, we outperformed SILK ran in parallel on 16 processors.

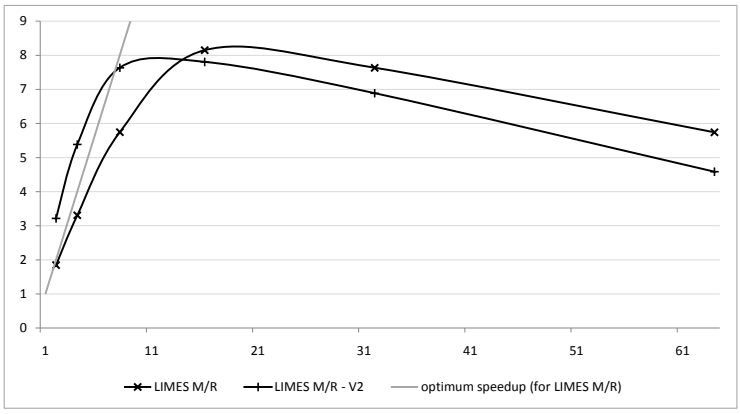
In future work, we will aim to develop an exemplar computation approach that takes the skew in the data distribution into consideration and thus enable a better load balancing across the Hadoop slaves. Furthermore, we will investigate new algorithms for Link Discovery and aim to parallelize them so as to enable LIMES to run on problems of complexity beyond 10^9 in less that a minute.



(a) Speedup graph – Diseases

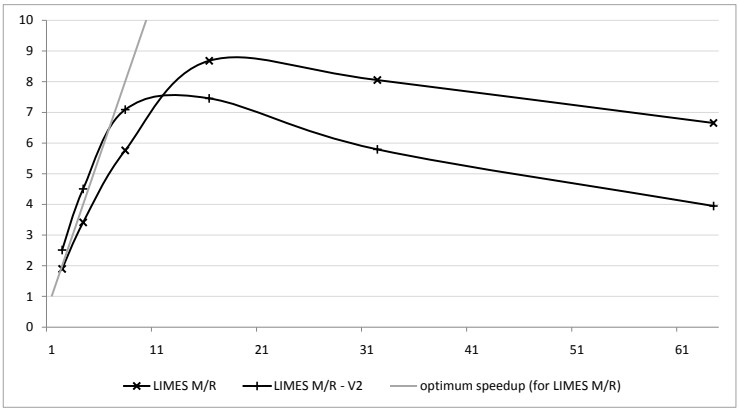


(b) Speedup graph – Similar Cities

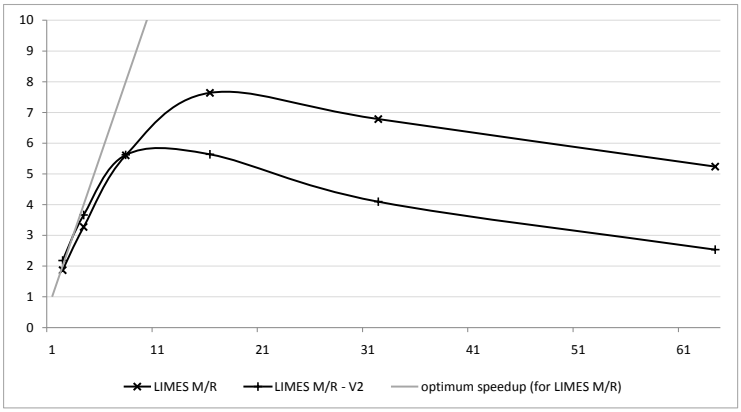


(c) Speedup graph – Similar Films

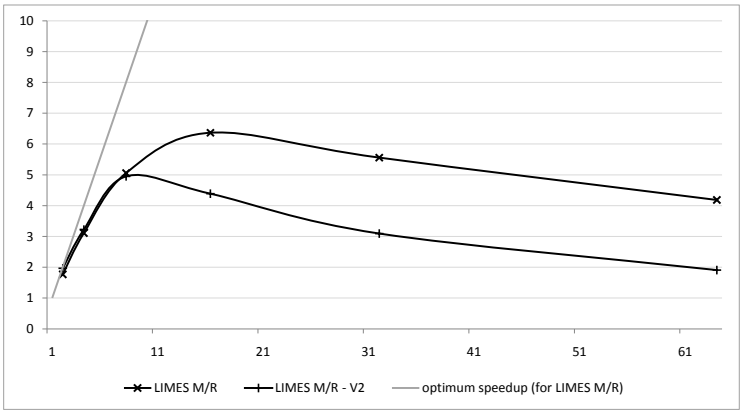
Figure 9.4: Experimental results for the first series of experiments. We display the runtime of SILK when using the full potential of our architecture.



(a) Threshold = 0.5



(b) Threshold = 0.9



(c) Threshold = 0.95

Figure 9.5: Experimental results for the second series of experiments

10 WHEN TO REACH FOR THE CLOUD

PREAMBLE

This section concludes the first part of this work and delves into the rapid execution of link specifications on different types of hardware. The chapter is taken from (Ngonga Ngomo et al., 2013) and was a joint work with the database department at the University of Leipzig. The author developed and implemented the core algorithm $\mathcal{HR}^3$ (see Chapter 5), co-designed the study and evaluation as well as co-wrote the paper.

10.1 INTRODUCTION

Link Discovery (LD) is of central importance for realizing the fourth Linked Data principle (Auer et al., 2013a; Ngonga Ngomo et al., 2014). With the growth of the Web of Data, the complexity of LD problems has grown considerably. For example, linking places from *LinkedGeoData* and *DBpedia* requires the comparison of hundreds of thousands of instances. Over the last years, several time-efficient algorithms such as *LIMES* (Ngonga Ngomo and Auer, 2011), *MultiBlock* (Isele et al., 2011) and $\mathcal{HR}^3$ (Ngonga Ngomo, 2012b) have been developed to address the problem of the a-priori quadratic runtime of LD approaches. In general, these algorithms aim at minimizing the number of unnecessary similarity computations to carry out. While these approaches have been shown to outperform naïve LD implementations by several orders of magnitude, the sheer size of the number of links can still lead to unpractical runtimes. Thus, cloud implementations of some of these algorithms (e.g., *LIMESMR* (Hillner and Ngonga Ngomo, 2011) and Silk MapReduce[1]) have been recently developed. The speed-up of these implementations is, however, limited by a considerable input-output overhead that can lead to worse runtimes than on single machines. Interestingly, the use of standard parallel hardware has recently been shown to have the potential to outperform cloud computing techniques (Heino and Pan, 2012).

The multiplicity of available hardware solutions for carrying out LD led us to ask the following fundamental question: *When should which type of hardware be used to optimize the runtime of LD processes?* Providing an answer to this question promises to enable the development of highly flexible and scalable LD frameworks that can adapt to the available hardware environment. It will allow to decide intelligently upon when to reach for remote computing services such as cloud computing services in contrast to using local resources such as graphics processing units (GPUs) or multi-processor and multi-core technology. To answer our research question, we compare the runtimes of several implementations of $\mathcal{HR}^3$ for several datasets and find break-even points for different hardware. We chose the $\mathcal{HR}^3$ algorithm because it is the first algorithm with a guaranteed reduction ratio (Ngonga Ngomo, 2012b). Thus, it promises to generate less overhead than other LD algorithms for comparable problems. Moreover, this algorithm can be used in manifold scenarios

1 https://www.assembla.com/spaces/silk/wiki/Silk_MapReduce

including LD, finding geographically related data (radial search) as well as search space reduction for other LD algorithms. The main contributions of this work are:

- We present the first implementation of a LD approach for GPUs. It relies on the GPU for fast parallel indexing and on the CPU for the computation of distances.
- We show how load-balancing for Map-Reduce can be carried out for LD approaches in affine spaces.
- We obtain guidelines for the use of different parallel hardware for LD by the means of a comparative evaluation of different implementations on real-world datasets from the Linked Open Data Cloud.

The remainder of the paper is organized as follows: We begin by giving a brief overview of $\mathcal{HR}^3$ and other paradigms used in this work. In Section 10.3, we then show how $\mathcal{HR}^3$ must be altered to run on GPUs. Section 10.4 focuses on the Map-Reduce implementation of $\mathcal{HR}^3$ as well as the corresponding load balancing approach. Section 10.5 presents a comparison of the runtimes of the different implementations of $\mathcal{HR}^3$ and derives break-even points for the different types of hardware.[2] The subsequent section gives an overview of related work. Finally, Section 10.7 summarizes our findings and presents future work.

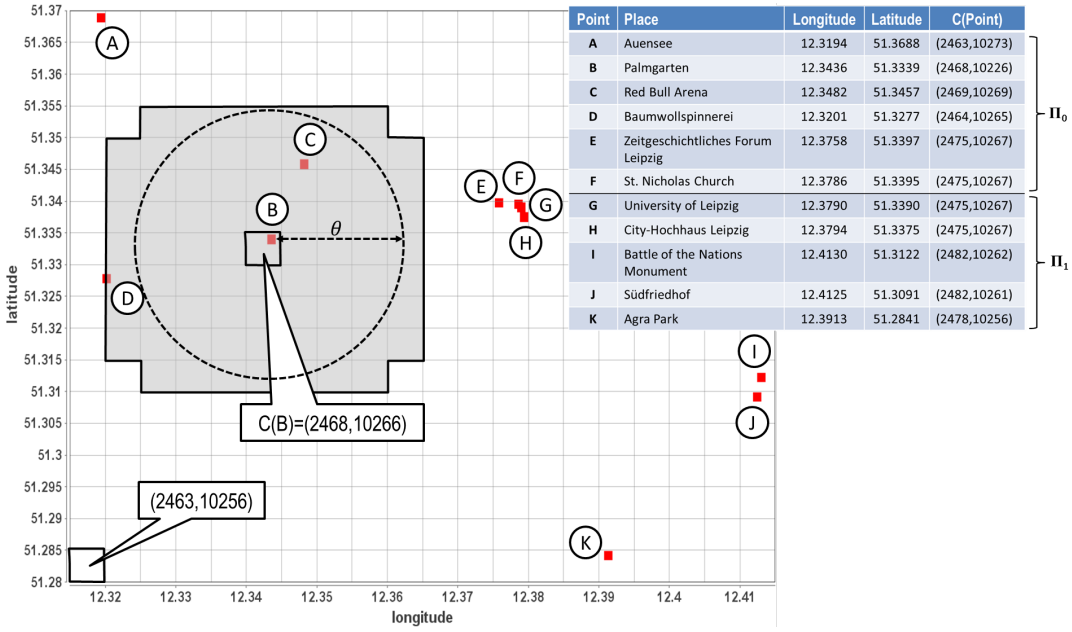
10.2 PRELIMINARIES

The specification of *link discovery* adopted herein is tantamount to the definition proposed in (Ngonga Ngomo, 2012b). Given a formal relation[3] R and two (not necessarily disjoint) sets of instances S and T, the goal of link discovery is to find the set $M = \{(s,t) \in S \times T : R(s,t)\}$. Given that the explicit computation of R is usually a very complex endeavour, most frameworks reduce the computation of M to that of the computation of an approximation $\tilde{M} = \{(s,t) : \delta(s,t) \leqslant \theta\}$, where δ is a (complex) distance function and θ is a distance threshold. Note that when $S = T$ and $R = \texttt{owl:sameAs}$, the link discovery task becomes a *deduplication* task. Naïve approaches to computing $\tilde{M}$ have a quadratic time complexity, which is impracticable on large datasets. Consequently, a large number of approaches has been developed to reduce this time complexity (see (Ngonga Ngomo, 2012b) for an overview). Most of these approaches achieve this goal by optimizing their reduction ratio. In newer literature, the $\mathcal{HR}^3$ algorithm (Ngonga Ngomo, 2012a) has been shown to be the first algorithm which guarantees that it can achieve any possible reduction ratio.

$\mathcal{HR}^3$ builds upon the HYPPO algorithm presented in (Ngonga Ngomo, 2011). The rationale of $\mathcal{HR}^3$ is to maximize the reduction ratio of the computation of $\tilde{M}$ in affine spaces with Minkowski measures. To achieve this goal, $\mathcal{HR}^3$ computes an approximation of $\tilde{M}$ within a discretization of the space $\Omega = S \cup T$. Each point $\omega = (\omega_1, \ldots, \omega_n) \in \Omega$ is mapped to discrete coordinates $(\lfloor \omega_1/\Delta \rfloor, \ldots, \lfloor \omega_n/\Delta \rfloor)$, where $\Delta = \theta/\alpha$ and $\alpha \in \mathbb{N} \backslash \{0\}$ is called the granularity parameter. An example of such a discretization is shown in Figure 10.1: The point B with coordinates $(12.3436, 51.3339)$ is mapped to the discrete coordinates $(2468, 10226)$. The set of all points with the same discrete coordinates forms a hypercube (short: cube) of

2 Details to the experiments and code are available at http://limes.sf.net.

3 For example, http://dbpedia.org/property/near



Point	Place	Longitude	Latitude	C(Point)	
A	Auensee	12.3194	51.3688	(2463,10273)	Π_0
B	Palmgarten	12.3436	51.3339	(2468,10226)	
C	Red Bull Arena	12.3482	51.3457	(2469,10269)	
D	Baumwollspinnerei	12.3201	51.3277	(2464,10265)	
E	Zeitgeschichtliches Forum Leipzig	12.3758	51.3397	(2475,10267)	
F	St. Nicholas Church	12.3786	51.3395	(2475,10267)	
G	University of Leipzig	12.3790	51.3390	(2475,10267)	Π_1
H	City-Hochhaus Leipzig	12.3794	51.3375	(2475,10267)	
I	Battle of the Nations Monument	12.4130	51.3122	(2482,10262)	
J	Südfriedhof	12.4125	51.3091	(2482,10261)	
K	Agra Park	12.3913	51.2841	(2478,10256)	

Figure 10.1: Example dataset containing 11 places from Leipzig. To identify all points with a maximum Euclidean distance $\theta = 0.02$, the space is virtually tiled into hypercubes with an edge length of $\Delta = \theta/4$. A cube is identified by its coordinates $(c_1, ..., c_n)$. The gray-shadowed cells indicate the cubes whose points are compared with B, i.e., $\{C' \mid index(C(B), C') \leqslant \alpha^p\}$.

width α in the space Ω. The cube that contains ω is called $C(\omega)$. We call the vector $(c_1, \ldots, c_n) = (\lfloor \omega_1/\Delta \rfloor, \ldots, \lfloor \omega_n/\Delta \rfloor) \in \mathbb{N}^n$ the coordinates of $C(\omega)$.

Given the distance threshold θ and the granularity parameter α, $\mathcal{HR}^3$ computes the set of candidates $t \in T$ for each $s \in S$ by using the index function given in Equation 10.1.

$$index(C, C') = \begin{cases} 0, \text{ if } \exists i : |c_i - c'_i| \leqslant 1 \text{ with } i \in \{1, ..., n\}, \\ \sum_{i=1}^{n} (|c_i - c'_i| - 1)^p \text{ else.} \end{cases} \tag{10.1}$$

where $C = C(s)$ and $C' = C(t)$ are hypercubes and p is the order of the Minkowski measure used in the space Ω.

Now, all source instances s are only compared with the target instances t such that $index(C(s), C(t)) \leqslant \alpha^p$. In our example, this is equivalent to computing the distance between B and all points contained in the gray-shadowed area on the map. Overall, $\mathcal{HR}^3$ achieve a reduction ratio of ≈ 0.82 on the data in Figure 10.1 as it only performs 10 comparisons instead of 55.

10.3 LINK DISCOVERY ON GPUS

10.3.1 *General-Purpose Computing on GPUs*

GPUs were originally developed for processing image data. Yet, they have been employed for general-purpose computing tasks in recent years. Compared to CPUs the architecture of GPU hardware exhibits a large number of simpler compute cores and is thus referred to as *massively parallel*. A single compute core typically

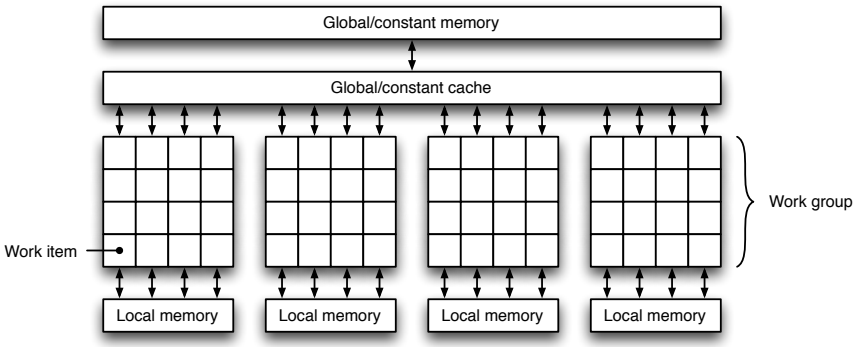


Figure 10.2: OpenCL memory model

contains several arithmetic and logic units (ALU) that execute the same instruction on multiple data streams (SIMD).

Parallel code on GPUs is written as *compute kernels*, the submission of which is orchestrated by a host program executed on the CPU. Several frameworks exist for performing general purpose computing on GPUs. In this work we use *OpenCL*[4], a vendor-agnostic industry standard. The memory model as exposed to OpenCL kernels is depicted in Figure 10.2: An instance of a compute kernel running on a device is called a *work item* or simply thread.[5] Work items are combined into *work groups*. All items within the same group have access to low-latency local memory and the ability to synchronize load/store operations using barriers. Thus, the actual number of kernel instances running in parallel is often limited by register and local memory usage. Each work item is assigned a globally (among all work items) and locally (within a work group) unique identifier, which also imposes a scheduling order. Typically those identifiers are used to compute local and global memory offsets for loading and storing data items that a given thread works on. Data transfer between host program and compute device is done via global device memory to which all work items have access, albeit with higher latency.

Threads on modern GPUs do not run in isolation. They are scheduled in groups of 64 or 32 work items depending on the hardware vendor. All threads within such a group execute the same instruction in lock-step. Any code path deviations due to control flow statements need to be executed by all items, throwing away unnecessary results (predication). It is therefore essential that each work item in such a group performs the same amount of work. The OpenCL framework does not expose the size of such groups to the API user. An upper bound is given by the work group size, which is always an integer multiple of the schedule group size.

10.3.2 *GPU-based $\mathcal{HR}^3$ Implementation*

For GPU-based computation all data must be copied to the device via the PCIe bus. We therefore only perform expensive computations on the device that benefit from the massive parallelism. In the case of $\mathcal{HR}^3$ this is the computation of the index function that determines which hypercubes a given cubes needs to be compared with. Since GPUs work best with regular memory access patterns a few preparation steps are needed. These are performed serially on the host. First, we

4 http://www.khronos.org/opencl/

5 We use the therms work item and thread interchangeably in this work.

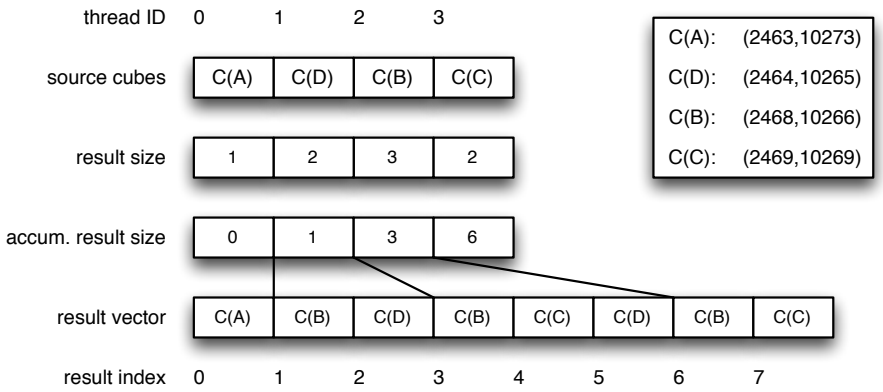


Figure 10.3: Result of the index computation on GPU hardware

discretize the input space $\Omega = S \cup T$, resulting in a set of hypercubes. All hypercubes are then sorted component-wise. The number of hypercubes determines the global work size. That is, each thread is assigned a hypercube (called *pivot cube*) determined by its global id. The work to be done by each thread is then to compute all those hypercubes that abide by the bound on indexes set by $\mathcal{HR}^3$.

A naÃŕve implementation would have each thread compare its pivot cube to all other cubes, resulting in an amount of work quadratic in the number of hypercubes. A better approach is to minimize the amount of cube comparisons while maintaining an even work distribution among threads within the same group. Since hypercubes are globally sorted and fetched by work items in increasing schedule order, the ordering is maintained also locally. That is, let $g = k + 1$ be the local work group size. The work item with the least local id per group is assigned the smallest pivot cube C^0 while the last work item having the highest local id operates on the largest cube C^k as its pivot. Both work items therefore can determine a lower and upper bound for the whole group as follows. The first item computes the cube $C^{0-\alpha} = (c_1^0 - \alpha, \ldots, c_n^0 - \alpha)$ and the last item computes the cube $C^{k+\alpha} = (c_1^k + \alpha, \ldots, c_n^k + \alpha)$, where c_i^0 and c_i^k are the coordinates of the respective pivot cubes. Thread 0 then determines i_l, the index of the largest cube not greater then $C^{0-\alpha}$ while thread k computes i_u, the index of the smallest cube that is greater than $C^{k+\alpha}$. After a barrier synchronization that ensures all work items in a group can read the values stored by threads 0 and k, all work items compare their pivot cube to cubes at indices $i_l, \ldots (i_u - 1)$ in global device memory. Since all work items access the same memory locations fetches can be efficiently served from global memory cache.

In OpenCL kernels dynamic memory management is not available. That is, all buffers used during a computation must be allocated in advance by the host program. In particular, the size of the result buffer must be known before submitting a kernel to a device. We therefore cannot simply write the resulting cubes to an output vector. Instead, we compute results in two passes. During the first pass each thread writes the number of results it needs to produce to an output vector. A prefix sum over this vector yields at each index the accumulated number of results of threads with a lower id. This value can be used as an index into the final output vector at which each thread can start writing its results.

As an example consider Figure 10.3. It shows four threads $(0 \ldots 3)$, each of which loads a single cube from the sorted *source cubes vector*. The index from which each

threads loads its cube is given by its id.[6] In this example we assume a granularity factor of $\alpha = 4$. For thread 1 the smallest cube its pivot cube needs to be compared with is $C(D) = (2464, 10265)$ while the largest is $C(B) = (2468, 10266)$. It therefore writes 2 into an output vector, again using its thread id as an index. Thread 0 as well as 2 and 3 do the same, which results in the *result size vector* as depicted in Figure 10.3. In order to determine the final indexes each thread can use for storing its results in the result vector, an exclusive prefix sum is computed over the result size vector. This operation computes at each index i the sum of the elements at indexes $0 \dots (i-1)$, resulting in the *accumulated result size vector*. A result vector of the appropriate size is allocated and in a second kernel run each thread can now write the cube coordinates starting at the index computed in the previous step. Indexing results are then copied back to the host where comparison of the actual input points is carried out. Since this operation is dominated by the construction of the result it cannot be significantly improved on parallel hardware.

10.4 MAPREDUCE-BASED LINK DISCOVERY

In this section we present an implementation of $\mathcal{HR}^3$ with MapReduce (MR), a programming model designed for parallelizing data-intensive computing in cluster environments (Dean and Ghemawat, 2008). MR implementations like *Apache Hadoop* rely on a distributed file system (DFS) that can be accessed by all nodes. Data is represented by key-value pairs and a computation is expressed employing two user-defined functions, `map` and `reduce`, which are processed by a fixed number of map (m) and reduce tasks (r). For each intermediate key-value pair produced in the map phase, a target reduce task is determined by applying a partitioning function that operates on the pair's key. The reduce tasks first sort incoming pairs by their intermediate keys. The sorted pairs are then grouped and the reduce function is invoked on all adjacent pairs of the same group.

We describe a straightforward realization of $\mathcal{HR}^3$ as well as an advanced approach that considers skew handling to guarantee load balancing and to avoid unnecessary data replication. In favor of readability, we consider a single dataset only.

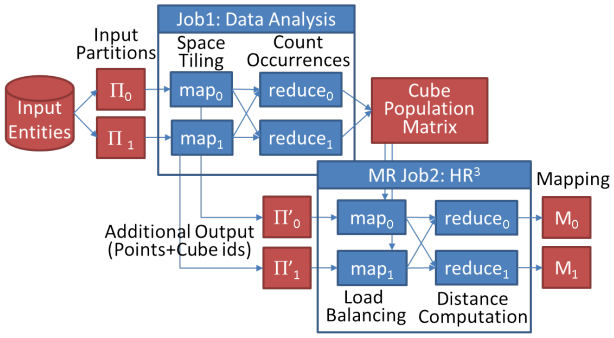
10.4.1 $\mathcal{HR}^3$ *with MapReduce*

$\mathcal{HR}^3$ can be implemented with a single MR job. The main idea is to compare the points of two related cubes within a single reduce call. We call two cubes C, C' *related* iff $\mathrm{index}(C, C') \leqslant \alpha^p$. For each input point ω, the map function determines the surrounding cube $C(\omega)$ and the set of related cubes RC, which might contain points within the maximum distance. For each cube $C' \in RC$, map outputs a $(\mathsf{cid}_1 \odot \mathsf{cid}_2 \odot \mathsf{flag}, (p, \mathsf{flag}))$ pair with a composite key and the point itself as value. The first two components of the key identify the two involved cubes using textual cube ids: $\mathsf{cid}_1 = \min\{C(\omega).\mathrm{id}, C'.\mathrm{id}\}$ and $\mathsf{cid}_2 = \max\{C(\omega).\mathrm{id}, C'.\mathrm{id}\}$. The flag indicates whether ω belongs to the first or to the second cube. The repartitioning of the output key-value pairs is done by applying a hash function on the first two key components. This assigns all points of $C(\omega) \cup C'$ to the same reduce task. All key-value pairs are sorted by their complete keys. Finally, the reduce function is in-

6 For means of readability we show only one id per thread that serves as both its local and global id.

Algorithm 10.1 Basic $\mathcal{HR}^3$- Map

Require: k_{in}=unused, $v_{in} = \omega$
1: $\Delta \leftarrow \theta/\alpha$
2: $cid_1 \leftarrow$ `getCubeId`$(C(\omega))$
3: $RC \leftarrow$ `getRelatedCubes`$(C(\omega), \Delta)$
4: **for all** $C' \in RC$ **do**
5: $cid_2 \leftarrow$ `getCubeId`(C')
6: **if** $cid_1 \leqslant cid_2$ **then**
7: `output`$(cid_1.cid_2.0, (\omega, 0))$
8: **else**
9: `output`$(cid_2.cid_1.1, (\omega, 1))$
10: **end if**
11: **end for**
`//part = hash(`cid_1`,`cid_2`) mod r`
`//sort component-wise by entire key`
`//group by` cid_1, cid_2



(a) $\mathcal{HR}^3$ with Load Balancing for MR

	Cube index	Partition Π_0	Partition Π_1
Cubes C_0	(2463,10273)	1	0
C_1	(2468,10226)	1	0
C_2	(2469,10269)	1	0
C_3	(2464,10265)	1	0
C_4	(2475,10267)	2	2
C_5	(2482,10262)	0	1
C_6	(2482,10261)	0	1
C_7	(2478,10256)	0	1

(b) CPM

Figure 10.4: Overview of the MR-based implementation with load balancing (left) and the cube population matrix for the example dataset with $m = 2$ (right)

voked on all values whose first two key components are equal. In reduce, the actual distance computation takes place. Due to the sorting, it is ensured that all points of the cube with the smaller cube id are processed first allowing for an efficient comparison of points of different cubes. The pseudo-code of the $\mathcal{HR}^3$ implementation is shown in Algorithm 10.1 and Algorithm 10.2.

The described approach has two major drawbacks. First, a map task operates only on a fraction of the input data without global knowledge about the overall data distribution. Thus, each point is replicated and repartitioned $|RC|$ times, independently of whether there are points in the related cubes or not. Second, this approach is vulnerable to data skew, i.e., due to the inherent quadratic time complexity varying cube sizes can lead to severe load imbalances of the reduce tasks. Depending on the problem size and the granularity of the space tiling, the scalability of the described approach might be limited to a few nodes only. We provide an advanced approach that addresses these drawbacks in the next section.

Algorithm 10.2 Basic $\mathcal{HR}^3$- Reduce

Require: k_{tmp}=cid_1, cid_2
Require: v_{tmp}=list< $\omega, flag$ >
1: $buf \leftarrow \{\}$
2: **if** $cid_1 = cid_2$ **then**
3: **for all** $(\omega, flag) \in v_{tmp}$ **do**
4: **for all** $\omega' \in buf$ **do**
5: compare(ω, ω')
6: **end for**
7: **end for**
8: $buf \leftarrow buf \cup \{\omega\}$
9: **else**
10: **for all** $(\omega, flag) \in v_{tmp}$ **do**
11: **if** flag=0 **then**
12: $buf \leftarrow buf \cup \{\omega\}$
13: **else**
14: **for all** $\omega' \in buf$ **do**
15: compare(ω, ω')
16: **end for**
17: **end if**
18: **end for**
19: **end if**

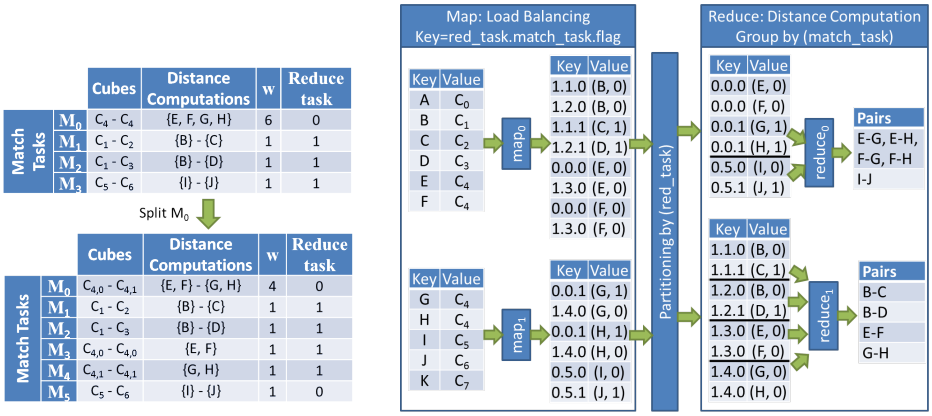
10.4.2 $\mathcal{HR}^3$ *with Load Balancing*

The advanced approach borrows ideas from the load balancing approaches for Entity Resolution presented in (Kolb et al., 2012b). An overview is shown in Figure 10.4a. The overall idea is to schedule a light-weight analysis MR job that linearly scans the input data in parallel and collects global data statistics. The second MR job utilizes these statistics for a data-driven redistribution of points ensuring evenly loaded reduce tasks.

Data Analysis Job. The first job calculates the cube index of each point in the map phase and sums up the number of points per (non-empty) cube in reduce. The output is a cube population matrix (CPM) of size $c \times m$ that specifies the number of points of c cubes across m input partitions. For our running example, an analysis job with $m = 2$ map tasks would read data from two input partitions Π_0 and Π_1 (cf. table in Figure 10.1) and produce the CPM shown in Figure 10.4b.

Distance Computation Job. The second MR job is based on the same number of map tasks and the same partitioning of the input data. At initialization, each map task reads the CPM. Similar to the basic approach, the reduce function processes pairs of related cubes. Because the CPM allows for an easy identification of empty cubes, the number of intermediate key-value pairs can be reduced significantly. As an example, for point B of the running example, the map function of the basic approach would output 77 key-value pairs. With the knowledge encoded in the CPM, this can be reduced to two pairs only, i.e., for computing B's distances to the points C and D, respectively.

Before processing the first input point, each map tasks constructs a list of so-called match tasks. A match task is a triple (C_i, C_j, w), where C_i, C_j are two re-



(a) (b)

Figure 10.5: Match task creation and reduce task assignment with/without splitting of large tasks (left). Example data flow for second MR job (right)

lated cubes and $w = |C_i| \cdot |C_j|$ ($w = |C_i| \cdot (|C_i| - 1)/2$ for $i = j$) is the corresponding workload. The overall workload W is the sum of the workload of all match tasks. To determine each match task's target reduce task, the list is sorted in descending order of the workload. In this order, match tasks are assigned to the r reduce tasks following a greedy heuristic, i.e., the current match task is assigned to the reduce task with the currently lowest overall workload. The resulting match tasks are shown on the top of Figure 10.5a. Obviously, the reduce tasks are still unevenly loaded, because a major part of the overall workload is made up by the match task $C_4 - C_4$. To address this, for each large match task $M = (C_i, C_j, w)$ with $w > W/r$, both cubes are split according to their input partitioning into m subcubes. Consequently, M is split into a set of smaller subtasks, each comprising a pair of split subcubes before the sorting and reduce task assignment takes place. The bottom of Figure 10.5a illustrates the splitting of the large match task $(C_4 - C_4)$. Because its workload $w = 6$ exceeds the average reduce task workload of $9/2 = 4.5$, C_4 is split into two subcubes $C_{4,0}$ (containing E, F) and $C_{4,1}$ (containing G, H). This results in three subtasks $(C_{4,0}, C_{4,0}, 1)$, $(C_{4,1}, C_{4,1}, 1)$, and $(C_{4,0}, C_{4,1}, 4)$ that recompose the original match task. Thus, both reduce tasks compute approximately the same number of distances indicating a good load balancing for the example.

After the initial match task creation, map task i builds an index that maps a cube to a set of corresponding match tasks. Thereby, only cubes of whom the input partition i actually contains points, need to be considered. For each input point ω and each match task of the cube $C(\omega)$, the map function outputs a $(\mathsf{red\_task} \odot \mathsf{match\_task} \odot \mathsf{flag}, (!, \mathsf{flag}))$ pair. Again, the flag indicates to which of the match task's (possibly split) cubes ω belongs to. The partitioning is only based on the reduce task index. The sorting is performed on the entire key, whereas the grouping is done by match task index. Figure 10.5b illustrates the dataflow for the running example. Note, that due to the enumeration of the match tasks and the sorting behavior, it is ensured that the largest match tasks are processed first. This makes it unlikely that larger delays occur at the end of the computation when most nodes are already idle.

Dataset	Source	Size	Features
DS_1	DBPedia	25,781	min/medium/max elevation
DS_2	DBPedia	475,000	latitude, longitude
DS_3	Linked Geo Data	500,000	latitude, longitude
DS_4	Linked Geo Data	6,000,000	latitude, longitude

Figure 10.6: Datasets used for evaluation

10.5 EVALUATION

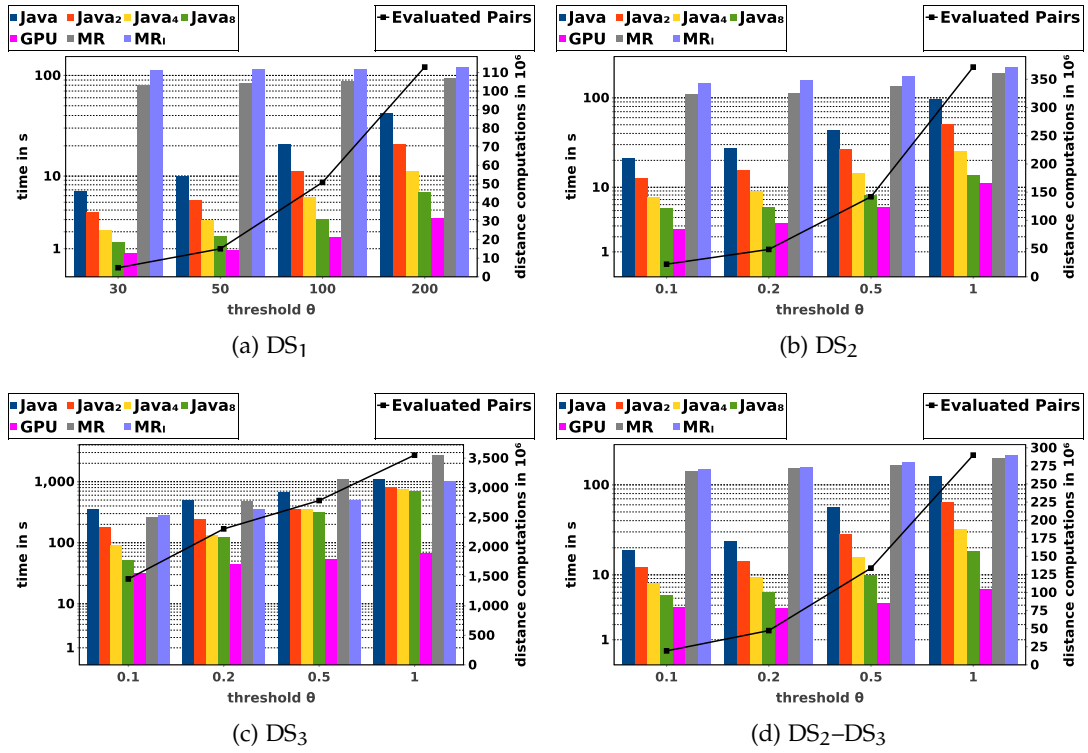
The aim of our evaluation was to discover break-even points for the use of parallel processor, GPU and cloud implementations of LD algorithms. For this purpose, we compared the runtimes of the implementations of $\mathcal{HR}^3$ presented in the previous sections on four datasets within two series of experiments. The goal of the first series of experiment was to compare the performance of the approaches for link discovery problems of common size. Thereafter, we carried out a scalability evaluation on a large dataset to detect break-even points of the implementations. In the following, we present the datasets we used as well as the results achieved by the different implementations.

10.5.1 *Experimental Setup*

We utilized the four datasets of different sizes shown in Figure 10.6. The small dataset DS_1 contains place instances having three elevation features. The medium-sized datasets DS_2 and DS_3 contain instances with geographic coordinates. For the scalability experiment we used the large dataset DS_3 and varied its size up to $6 \cdot 10^6$. Throughout all experiments we considered the Euclidean distance. Given the spectrum of implementations at hand, we ran our experiments on three different platforms. The *CPU experiments* (Java, Java_2, Java_4, Java_8 for 1, 2, 4 and 8 cores) were carried out on a 32-core server running JDK 1.7 on Linux 10.04. The processors were 8 quad core AMD Opteron 6128 clocked at 2.0 GHz. The *GPU experiments* (GPU) were performed on an average consumer workstation. The GPU was a AMD Radeon 7870 GPU with 20 compute units, each of which has the ability to schedule up to 64 parallel hardware threads. The host program was executed on a Linux workstation running Ubuntu 12.10 and AMD APP SDK 2.8. The machine had an Intel Core i7 3770 CPU and 8 GB of RAM. All C++ code was compiled with gcc 4.7.2. Given that C++ and Java are optimized differently, we also ran the Java code on this machine and computed a runtime ratio that allowed our results to remain compatible. The *MapReduce experiments* (basic: MR, load balanced: MR_1) were performed with the *Dedoop prototype* (Kolb et al., 2012a) on Amazon EC2 in EU-west location. For the first experiment we used 10 nodes of type c1.medium (2 virtual cores, 1.7 GB memory). For the large dataset we employed 20 nodes of type c1.xlarge (8 virtual cores, 7 GB memory).

10.5.2 *Performance Comparison*

The results of our performance comparison are shown in Figure 10.7. While the parallel implementation of $\mathcal{HR}^3$ on CPUs scales linearly for uniformly distributed



(a) DS_1 (b) DS_2

(c) DS_3 (d) DS_2–DS_3

Figure 10.7: Comparison of runtimes for Experiment 1

data, the considerable skew in the DS_3 data led to the 8-core version being only 1.6 times faster than the mono-core implementation with a threshold of 1°. This impressively demonstrates the need for load balancing in all parallel link discovery tasks on skewed data. This need is further justified by the results achieved by MR and MR_l on DS_3. Here, MR_l clearly outperforms MR and is up to 2.7 times faster. Still, the most important result of this series of experiments becomes evident after taking a look at the GPU and Java runtimes on the workstation.

Most importantly, the massively parallel implementation outperforms all other implementations significantly. Especially, the GPU implementation outperforms the MR and MR_l by one to two orders of magnitude. Even the $Java_8$ implementation is outperformed by up to one order of magnitude. The performance boost of the GPU is partly due to the different hardware used in the experiments. To measure the effect of the hardware, we ran the server Java program also on the workstation. A comparison of the runtimes achieved during this rerun shows that the workstation is between 2.16 and 7.36 times faster than the server. Still, our results suggests that our massively parallel implementation can make an effective use of the underlying architecture to outperform all other implementations in the indexing phase. The added efficient implementation of float operations for the distance computation in C++ leads to an overall superior performance of the GPU. Here, the results can be regarded as conclusive with respect to MR and MR_l and clearly suggest the use of local parallelism when dealing with small to average-sized link discovery problems.

The key observation that leads to conclusive results when comparing GPU and CPU results is that the generation of the cube index required between 29.3% (DS_1, $\theta = 50m$) and 74.5% (DS_3, $\theta = 1°$) of the total runtime of the algorithm during

the deduplication tasks. Consequently, while running a parallel implementation on the CPU is advisable for small datasets with small thresholds for which the index computation makes up a small percentage of the total computation, running the approach on medium-sized datasets or with larger thresholds should be carried out on the GPU. This conclusion is yet only valid as long as the index fits into the memory of the GPU, which is in most cases 4 to 8 times smaller than the main memory of workstations. Medium-sized link discovery tasks that do not fit in the GPU memory should indeed be carried out on the CPUs. Our experiments suggest a break-even point between CPU and GPU for result set sizes around 10^8 pairs for 2-dimensional data. For higher-dimensional data where the index computation is more expensive, the break-even point is reached even for problems smaller than DS_1.

10.5.3 *Scalability: Data Size*

The strengths of the cloud are revealed in the second series of experiments we performed (see Figure 10.8). While the DFS and data transfer overhead dominates the total runtime of the LD tasks on the small datasets, running the scalability experiments on 20 nodes reveals that for tasks which generate more than 12 billion pairs as output, MR_l outperforms our local Java implementation. Moreover, we ran further experiments with more than 20 nodes on the 6 million data items. Due to its good scalability, the cloud implementation achieves the runtime of the GPU or performs even better for more nodes, e.g., for 30 (50) nodes MR_l requires approx. 32min (23min). It is important to remember here that the GPU implementation runs the comparisons in the CPU(s). Thus, the above suggested break-even point will clearly be reached for even smaller dataset sizes with more complex similarity measures such as the Levenshtein distance or the trigram similarity. Overall, our results hint towards the use of local massively parallel hardware being sufficient for a large number of link discovery tasks that seemed to require cloud infrastructures. Especially, numeric datasets can be easily processed locally as they require less memory than datasets in which strings play the central role. Still, for LD tasks whose intermediate results go beyond 10^{10} pairs, the use of the cloud still remains the most practicable solution. The clue for deciding which approach to use lies in having an accurate approximation function for the size of the intermediate results. $\mathcal{HR}^3$ provides such a function and can ensure that it can achieve an approximation below or equal to any possible error margin. Providing such guarantees for other algorithms would thus allow deciding effectively and conclusively when to reach for the cloud.

10.6 RELATED WORK

Link discovery has become an important area of research over the last few years. Herein, we present a brief overview of existing approaches.[7] Overall, the two main problems time complexity and generation of link specifications have been at the core of the research on LD.

With regard to *time complexity*, time-efficient string comparison algorithms such as PPJoin+ (Xiao et al., 2008), EDJoin (Xiao et al., 2008) that were developed for deduplication were integrated into several link discovery frameworks such as

7 See (Ngonga Ngomo, 2012b; Isele et al., 2012) for more extensive presentations of the state of the art.

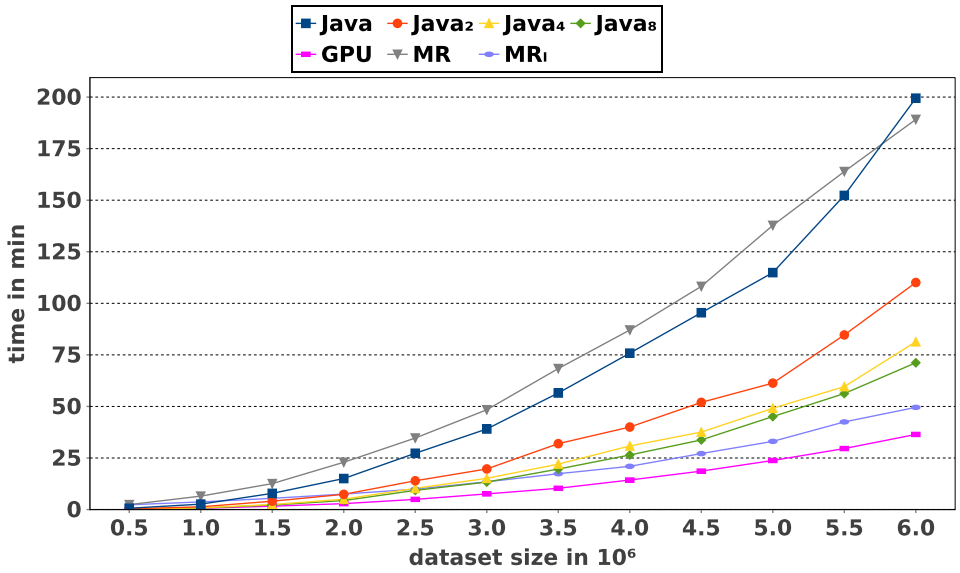


Figure 10.8: Comparison of runtimes on DS4

LIMES (Ngonga Ngomo, 2012b). Moreover, dedicated time-efficient approaches were developed for LD. For example in (Ngonga Ngomo and Auer, 2011), an approach based on the Cauchy-Schwarz inequality is presented. The approaches HYPPO (Ngonga Ngomo, 2011) and $\mathcal{HR}^3$ (Ngonga Ngomo, 2012a) rely on space tiling in spaces with measures that can be split into independent measures across the dimensions of the problem at hand. Especially, $\mathcal{HR}^3$ was shown to be the first approach that can achieve a relative reduction ratio r' less or equal to any given relative reduction ratio $r > 1$. Standard blocking approaches were implemented in the first versions of SILK and later replaced with MultiBlock (Isele et al., 2011), a lossless multi-dimensional blocking technique. KnoFuss (Nikolov et al., 2012) also implements blocking techniques to achieve acceptable runtimes. Further LD frameworks have been participated in the ontology alignment evaluation initiative (Euzenat et al., 2011).

With regard to the *generation of link specifications*, some unsupervised techniques were newly developed (see, e.g., (Nikolov et al., 2012)), but most of the approaches developed so far abide by the paradigm of supervised machine learning. For example, the approach presented in (Isele and Bizer, 2011) relies on large amounts of training data to detect accurate link specification using genetic programming. RAVEN (Ngonga Ngomo et al., 2011) is (to the best of our knowledge) the first active learning technique for LD. The approach was implemented for linear or Boolean classifiers and shown to require a small number of queries to achieve high accuracy. Later, approaches combining active learning and genetic programming for LD were developed (Isele and Bizer, 2012; Ngonga Ngomo and Lyko, 2012).

The entity resolution (ER) problem (see (Elmagarmid et al., 2007; Köpcke et al., 2010) for surveys) shares many similarities with link discovery. The MR programming model has been successfully applied for both ER and LD. (Vernica et al., 2010) proposes a MR implementation of the PPJoin+ algorithm for large datasets. A first application for MR-based duplicate detection was presented in (Wang et al., 2010). In addition, (Hillner and Ngonga Ngomo, 2011) as well as *Silk MapReduce*[8] imple-

8 https://www.assembla.com/spaces/silk/wiki/Silk_MapReduce

ment MR approaches for LD. Several MR implementations for blocking-based ER approaches have been investigated so far. An MR implementation of the popular sorted neighbourhood strategy is presented in (Kolb et al., 2012c). Load balancing for clustering-based similarity computation with MR was considered in (Kolb et al., 2012b). The ER framework Dedoop (Kolb et al., 2012a) allows to specify advanced ER strategies that are transformed to executable MR workflows and submitted to Hadoop clusters.

Load balancing and skew handling are well-known problems for parallel data processing but have only recently gained attention for MapReduce. *SkewTune* (Kwon et al., 2012) is a generic load balancing approach that is invoked for a MapReduce job as soon as the first map (reduce) process becomes idle and no more map (reduce) tasks are pending. Then, the remaining keys (keygroups) of running tasks are tried to redistribute so that the capacity of the idle nodes is utilized. The approach in (Gufler et al., 2012) is similar to our previous load balancing work (Kolb et al., 2012b) as it also relies on cardinality estimates determined during the map phase of the computation.

10.7 CONCLUSION AND FUTURE WORK

In this paper, we presented a comparison of the runtimes of various implementations of the same link discovery approach on different types of parallel hardware. In particular, we compare parallel CPU, GPU and MR implementations of the $\mathcal{HR}^3$ algorithm. Our results show that the CPU implementation is most viable for two-dimensional problems whose result set size is in the order of 10^8. For higher-dimensional problems, massively parallel hardware preforms best even for problem with results set sizes in the order of 10^6. Cloud implementations become particularly viable as soon as the result set sizes reach the order of 10^{10}. Our results demonstrate that efficient resource management for link discovery demands the development of accurate approaches for determining the size of the intermediate results of link discovery frameworks. $\mathcal{HR}^3$ provides such a function. Thus, in future work, we will aim at developing such approximations for string-based algorithms. Moreover, we will apply the results presented herein to develop link discovery approaches that can make flexible use of the hardware landscape in which they are embedded.

Part III

LEARNING LINK SPECIFICATIONS

In the third part of this work, we focus on the second challenge behind Link Discovery, i.e., the accuracy problem. We present a series of approaches that address this problem by using different machine-learning paradigms, including active learning and unsupervised learning. First, we present a brief comparison of commonly used machine learning approaches on the link discovery task (Chapter 16). Thereafter, we focus mainly on active learning (Chapter 11–Chapter 13). We then delve into unsupervised learning (Chapter 14–Chapter 15) and conclude this part with an approach that aims to support learning by providing keys for link discovery (Chapter 17).

11

RAVEN: RAPID ACTIVE LEARNING OF LINK SPECIFICATIONS

PREAMBLE

This chapter presents the first active learning approach dedicated to link discovery (to the best of our knowledge). It is an extended version of Ngonga Ngomo et al. (2011). The author designed the algorithm, implemented it and carried out the evaluation.

11.1 INTRODUCTION

The rationale of the Linked Data paradigm is to facilitate the transition from the document-oriented to the Semantic Web by extending the current Web with a commons of interlinked data sources (Bizer et al., 2009). Two key challenges arise when trying to discover links between data sources: the computational complexity of the matching task *per se* and the selection of an appropriate configuration for achieving maximal recall and precision.

The first challenge lies in the a-priori *complexity* of a matching task being proportional to the product of the number of instances in the source and target data source, an unpractical proposition as soon as the source and target knowledge bases become large. With the *LIMES framework*[1] (Ngonga Ngomo and Auer, 2011), we addressed this challenge by providing a lossless approach for time-efficient link discovery that is often an order of magnitude faster than other state-of-the-art tools.

The second challenge of the link discovery process lies in the specification of an appropriate *configuration* for the tool of choice. Such a specification usually consists of a set of restrictions on the source and target knowledge base, a list of properties of the source and target knowledge base to use for similarity detection, a combination of suitable similarity measures (e.g., Levenshtein (Levenshtein, 1966)) and similarity thresholds. Until now, such link discovery specifications are usually defined manually, in most cases via a time-consuming trial-and-error approach. Yet, the choice of a suitable configuration decides upon whether satisfactory links can be discovered. Specifying complex link configurations is a tedious process, as the user does not necessary know which combinations of properties lead to an accurate linking configuration. The difficulty of devising suitable link discovery specifications is amplified on the Web of Data by the *sheer size* of the knowledge bases (which often contain millions of instances) and their *heterogeneity* (i.e., by the complexity of the underlying ontologies, which can contain thousands of different types of instances and properties) (Bizer et al., 2009).

In this paper, we present the RApid actiVE liNking (RAVEN) approach. To the best of our knowledge, RAVEN is the first approach to apply active learning techniques for the semi-automatic detection of specifications for link discovery. Our approach is based on a combination of stable matching problems (as known from

1 http://limes.sf.net

machine learning) and a novel active learning algorithm derived from perceptron learning. RAVEN allows to determine:

- A sorted *matching of classes to interlink*; this matching represents the set of restrictions on the source and target knowledge bases.
- A stable *matching of properties* based on the selected restrictions that specifies the similarity space within which the linking is to be carried out.
- A highly accurate *link specification* including similarity measures and thresholds obtained via active learning.

Our evaluation with three series of experiments shows that we can compute linking configurations that achieve more than 90% F-score by asking the user to verify at most twelve potential links. RAVEN is *generic* enough to be employed with any link discovery framework that supports complex link specifications. The results presented herein were obtained using the LIMES framework for linking. We chose LIMES because it implements lossless approaches and is very time-efficient. A graphical user interface for the approach will be available within SAIM.[2]

This article is an extension of the corresponding workshop article presenting the RAVEN approach in (Ngonga Ngomo et al., 2011). Changes include an extended discussion of the evaluation results, e.g. the inclusion of property and class matching. Furthermore, the related work part was extended as well as further illustrations and explanations added throughout the paper.

After discussing related work in Section 11.2, we explain preliminary notions for link discovery and stable matching in Section 11.3. The approach itself is discussed in Section 11.4. We continue by analysing RAVEN with three different linking tasks in Section 11.5 and finally conclude in Section 11.6 with an outlook on future work.

11.2 RELATED WORK

Previous work related to this article can be divided in two main areas: the computation of links called *link discovery* and the *learning of link heuristics*.

11.2.1 *Link Discovery*

Current approaches for link discovery on the Web of Data can be subdivided into two categories: *domain-specific* and *universal* approaches.

Domain-specific link discovery frameworks aim at discovering links between knowledge bases from a particular domain. For example, the approach implemented in RKB knowledge base (RKB-CRS) (Glaser et al., 2009) focuses on computing links between universities and conferences while GNAT (Raimond et al., 2008) discovers links between music datasets. Further simple or domain-specific approaches can be found in (Cudré-Mauroux et al., 2009; Hogan et al., 2010; Nikolov et al., 2009; Papadakis et al., 2011; Sleeman and Finin, 2010; Volz et al., 2009).

Universal link discovery frameworks are designed to carry out mapping tasks independently from the domain of the source and target knowledge bases. For example, RDF-AI (Scharffe et al., 2009) implements a five-step approach that comprises the preprocessing, matching, fusion, interlinking and post-processing of datasets.

2 http://saim.sf.net

SILK (Volz et al., 2009) is a time-optimized tool for link discovery. It implements a multi-dimensional blocking approach that projects the instances to match in a multi-dimensional metric space. Subsequently, this space is subdivided into to overlapping blocks that are used to retrieve matching instances without loosing links. Another lossless Link Discovery framework is LIMES (Ngonga Ngomo and Auer, 2011), which addresses the scalability problem by utilizing the *triangle inequality* in metric spaces to compute pessimistic estimates of instance similarities. Based on these approximations, LIMES can filter out a large number of instance pairs that cannot suffice the matching condition set by the user. The experiments presented herein were carried out with LIMES.

The task of discovering links between knowledge bases is closely related with record linkage (Elmagarmid et al., 2007; Winkler, 2006) and deduplication (Bleiholder and Naumann, 2008). The database community has produced a vast amount of literature on efficient algorithms for solving these problems. Different blocking techniques such as standard blocking, sorted-neighborhood, bi-gram indexing, canopy clustering and adaptive blocking (see, e.g., (Köpcke et al., 2009)) have been developed to address the problem of the quadratic time complexity of brute force comparison methods.

11.2.2 *Learning Link Heuristics*

The second relevant research area for this paper is related to learning link specifications, which is usually carried out using a combination of shallow natural language processing (NLP) and machine learning methods. The existing methods aim at either one or both of the following goals: On the one hand, link creation should be made more reliable than purely manual approaches by using manual samples (supervised learning) for estimating precision and recall, user feedback (active learning) or analyzing network and other characteristics. On the other hand, those methods should also simplify the link creation process. As explained above, finding good interlinking heuristics can be burdensome and both non-experts and experts in the considered domain may struggle to find corresponding classes, properties, metrics and weights for their combination.

There has been a significant body of research work dedicated to matching ontologies (Granitzer et al., 2010; Leser and Naumann, 2007; Rahm and Bernstein, 2001; van Hage, 2008), including benchmarks in the ontology alignment evaluation initiative (OAEI). A recent comprehensive survey can be found in (Bellahsene et al., 2011), which covers many aspects of the research field. Finding links on instance level, which is the primary concern of this paper, has received less attention, although OAEI has been extended in 2009 with benchmarks in this area (Euzenat et al., 2010).

One approach in this area is RiMOM (Li et al., 2009), which combines several techniques to compute matchings. When matching instances, it takes the schema of the knowledge bases into account. RiMOM combines several strategies and similarity functions and works unsupervised, i.e. without training. Another approach is ObjectCoRef (Hu et al., 2010). It is based on learning the most distinctive features, i.e. property-value pairs, of entities in knowledge bases. In contrast to other approaches, it is not aimed at computing all links between two knowledge bases, but considers the task of linking entities in a whole cloud of knowledge bases (typ-

ically, the LOD cloud[3]) in a semi-supervised approach. Another recent approach is SERIMI (Araujo et al., 2011). It proceeds in two phases: a selection and a disambiguation phase. In the selection phase, SERIMI computes a mapping which interlinks entities in two input knowledge bases with low precision and high recall. In this phase, it relies on string similarity of the labels of entities. The disambiguation phase filters the output of the first phase by deciding amongst candidates with equal or similar labels.

In both cases, one of the problems is to obtain appropriate data for utilizing machine learning approaches without overburdening the user (Bilenko and Mooney, 2003; Köpcke and Rahm, 2008). For this reason, active learning has been employed by the database community (Arasu et al., 2010; Sarawagi and Bhamidipaty, 2002; Sarawagi et al., 2002). Active learning approaches usually present only few match candidates to the user for manual verification. The technique is particularly efficient in terms of required user input (Settles, 2009), because the user is only confronted with those match candidates which provide a high benefit for the underlying learning algorithm.

The RAVEN approach presented in this article goes beyond the state of the art in several ways: It is the first RDF-based approach to use machine learning for obtaining interlinking heuristics. In addition, it is the first active learning algorithm in this area. Moreover, it is the first approach to detect corresponding classes and properties automatically for the purpose of Link Discovery. Note that this challenge is very specific to and particularly relevant for the Data Web. In similar approaches developed for databases, the mapping of columns is often assumed to be known (Arasu et al., 2010). Yet, this assumption cannot be made when trying to link knowledge bases from the Web of Data because of the possible size of the underlying ontology. By supporting the automatic detection of links, we are able to handle heterogeneous knowledge bases with extremely large schemata.

11.3 PRELIMINARIES

Our approach to the active learning of linkage specifications extends ideas from several research areas, especially classification and stable matching problems. In the following, we present the notation that we use throughout this article and explain the theoretical framework underlying our work.

11.3.1 *Problem Definition*

The link discovery problem, which is similar to the record linkage problem, is an ill-defined problem and is consequently difficult to model formally (Arasu et al., 2010). In general, link discovery aims to discover pairs $(s, t) \in S \times T$ related via a relation R.

Definition 5 (Link Discovery)**.** *Given two sets* S *(source) and* T *(target) of entities, compute the set* $\mathcal{M}$ *of pairs of instances* $(s, t) \in S \times T$ *such that* $R(s, t)$.

The sets S resp. T are usually (not necessarily disjoint) subsets of the instances contained in two (not necessarily disjoint) knowledge bases $\mathcal{K}_S$ resp. $\mathcal{K}_T$. In most cases, the computation of whether $R(s, t)$ holds for two elements is carried out by projecting the elements of S and T based on their properties in a similarity space

3 http://lod-cloud.net

$\mathfrak{S}$ and setting $R(s,t)$ iff some similarity condition is satisfied. The specification of the sets S and T and of this similarity condition is usually performed within a *link specification* which is the input for a link discovery framework such as LIMES or SILK.

Definition 6 (Link Specification). *A link specification consists of three parts: (1) two sets of restrictions $\mathcal{R}_1^S$ ... $\mathcal{R}_m^S$ resp. $\mathcal{R}_1^T$ ... $\mathcal{R}_k^T$ that specify the sets S resp. T, (2) a specification of a complex similarity metric σ via the combination of several atomic similarity measures σ_1, ..., σ_n and (3) a set of thresholds τ_1, ..., τ_n such that τ_i is the threshold for σ_i.*

A restriction $\mathcal{R}$ is generally a logical predicate. Typically, restrictions in link specifications state (a) the `rdf:type` of the elements of the set they describe, i.e., $\mathcal{R}(x) \leftrightarrow x$ `rdf:type someClass` or (b) the features the elements of the set must have, e.g., $\mathcal{R}(x) \leftrightarrow (\exists y : x$ `someProperty` $y)$. Each $s \in S$ must abide by each of the restrictions $\mathcal{R}_1^S$ ... $\mathcal{R}_m^S$, while each $t \in T$ must abide by each of the restrictions $\mathcal{R}_1^T$ ... $\mathcal{R}_k^T$. Note that the atomic similarity functions $\sigma_1, ..., \sigma_n$ can be combined to σ by different means. In this paper, we will focus on using boolean operators and real weights combined as conjunctions. Also note that we are aware that several other categories of approaches can be used to determine pairs (s,t) such that $R(s,t)$, including approaches based on ontology matching, semantic similarity and formal inference. In this paper, we will be concerned exclusively with link discovery problems that can be specified via link specifications as defined above.

According to the formalizations of link discovery and link specifications above, finding matching pairs of entities can be defined equivalently as a classification task, where the classifier $\mathcal{C}$ maps each pair $(s,t) \in S \times T$ to one of the classes $\{-1, +1\}$.

Definition 7 (Link Discovery as Classification). *Given the set $S \times T$ of possible matches, the goal of link discovery is to find a classifier $\mathcal{C} : S \times T \rightarrow \{-1, +1\}$ such that $\mathcal{C}$ maps non-matches to the class -1 and matches to $+1$. $\mathcal{M}$ is then the set $\{(s,t) : \mathcal{C}(s,t) = +1\}$.*

In general, we assume classifiers that operate in an n-dimensional similarity space $\mathfrak{S}$. Each of the dimensions of $\mathfrak{S}$ is defined by a similarity function σ_i that operates on a certain pair of attributes of s and t. Each classifier $\mathcal{C}$ on $\mathfrak{S}$ can be modeled via a specific function $\mathcal{F}_\mathcal{C}$. $\mathcal{C}$ then returns $+1$ iff the logical statement $\mathcal{P}_\mathcal{C}(\mathcal{F}_\mathcal{C}(s,t))$ holds and -1 otherwise, where $\mathcal{P}_\mathcal{C}$ is what we call the specific predicate of $\mathcal{C}$. In this work, we consider two families of classifiers: *linear weighted* classifiers $\mathcal{L}$ and *boolean conjunctive* classifiers $\mathcal{B}$. The specific function of linear weighted classifiers is of the form

$$\mathcal{F}_\mathcal{L}(s,t) = \sum_{i=1}^{n} \omega_i \sigma_i(s,t), \tag{11.1}$$

where $\omega_i \in \mathbb{R}$. The predicate $\mathcal{P}_\mathcal{L}$ for a linear classifier is of the form $\mathcal{P}_\mathcal{L}(X) \leftrightarrow (X \geqslant \tau)$, where $\tau = \tau_1 = ... = \tau_n \in [0,1]$ is the similarity threshold. A boolean classifier $\mathcal{B}$ is a conjunction of n atomic linear classifiers $\mathcal{C}_1$, ... ,$\mathcal{C}_n$, i.e., a conjunction of classifiers that each operate on exactly one of the n dimensions of the similarity space $\mathfrak{S}$. Thus, the specific function $\mathcal{F}_\mathcal{B}$ is a boolean function of the form

$$\mathcal{F}_\mathcal{B}(s,t) = \bigwedge_{i=0}^{n} (\sigma_i(s,t) \geqslant \tau_i) \tag{11.2}$$

and the specific predicate is simply $\mathcal{P}_{\mathcal{B}}(X) = X$. Note, that given that classifiers are usually learned by using iterative approaches. We will denote classifiers, weights and thresholds at the t^{th} iteration by using superscripts, i.e., $\mathcal{C}^t$, ω_i^t and τ_i^t.

Current approaches to learning in record matching assume that the similarity space $\mathfrak{S}$ is given. While this is a sensible premise for mapping problems which rely on simple schemas, the large schemas (i.e., the ontologies) that underlie many datasets in the Web of Data do not allow such an assumption. DBpedia (Auer et al., 2007; Bizer et al., 2009; Stadler et al., 2010) (version 3.6) for example contains 289,016 classes which are partially mapped to 275 classes from the main DBpedia ontology. Moreover, it contains 42,016 properties, which are partially mapped to 1,335 properties from the main DBpedia ontology. Thus, it would be extremely challenging and tedious at best for a user to specify the properties to map when carrying out a simple deduplication analysis, let alone more complex tasks using the DBpedia dataset. Other datasets in the LOD cloud, such as LinkedGeoData (Auer et al., 2009; Stadler et al., 2012a) are even larger or have a similar size. Thus, being able to scale to those datasets is of crucial importance. In the following, we give a brief overview of stable matching problems, which we use to solve the problem of suggesting appropriate sets of restrictions on data and matching properties to generate a similarity space $\mathfrak{S}$ in which the link discovery problem can be carried out.

11.3.2 *Stable Matching Problems*

The best known stable matching problem is the stable marriage problem $\mathcal{SM}$ as formulated by (Gale and Shapley, 1962). The basic problem here is as follows: given two sets of males and females of equal magnitude, compute male-female pairings that are such that none of the partners p_1 in the pairings can cheat on his/her partner with another person p_2 that he/she prefers. Cheating is considered to be possible iff this other person, i.e., p_2, also considers the partner willing to cheat (p_1) more suitable than his/her current partner.

Formally, we assume two sets M (males) and F (females) such that $|M| = |F|$ and two functions $\mu : M \times F \to \{1, ..., |F|\}$ resp. $\gamma : M \times F \to \{1, ..., |M|\}$, that give the degree to which a male likes a female resp. a female a male. $\mu(m, f) > \mu(m, f')$ means that m prefers f to f'. Note, that for all f and f' where $f \neq f'$ holds, $\mu(m, f) \neq \mu(m, f')$ must also hold. Analogously, $m \neq m'$ implies $\gamma(m, f) \neq \gamma(m', f)$. A bijective function $s : M \to F$ is called a stable matching iff for all m, m', f, f' the following holds:

$$(s(m) = f) \wedge \big(s(m') = f'\big) \wedge \big(\mu(m, f') > \mu(m, f)\big) \to \big(\gamma(m', f') > \gamma(m, f')\big) \quad (11.3)$$

In (Gale and Shapley, 1962) an algorithm for solving this problem is presented and it is shown how it can be used to solve a generalization of the stable marriage problem, i.e., the well-know Hospital/Residents ($\mathcal{HR}$) problem. Formally, $\mathcal{HR}$ assumes a set R of residents $r \in R$ that each have a sorted preference list of $p(r)$ of hospitals they would like admission to. The list $p(r)$ can be derived from the preference function μ as defined for the stable marriage problem. We write $p(x, y) = n$ to state that y is at position n in the preference list of x, where x can be a hospital or a resident. Each hospital h from the set H of hospitals also has a preference list $p(h)$ of residents and a maximal capacity $c(h)$. Similarly to $p(r)$, the list $p(h)$ can be derived from the preference function γ as defined for the stable marriage problem.

Algorithm 11.1 RAVEN's stable matching algorithm

Require: Set of residents R
Require: Set of hospitals H
Require: Preference function p
$\mathcal{M} = \emptyset$ // Mapping of hospitals to residents
for $r \in R$ **do**
  $i(r) = 0$ //index for preference function
end for
for $h \in H$ **do**
  $c(h) = \left\lceil \frac{|R|}{|H|} \right\rceil$ //capacity setting
end for
while $R \neq \emptyset$ **do**
  for $r \in R$ **do**
    $h = p(r)[i(r)]$
    if $|\mathcal{M}(h)| < c(h)$ **then**
      $\mathcal{M}(h) := \mathcal{M}(h) \cup \{r\}$
      $R = R \backslash \{r\}$
    else
      if $\exists r' \in \mathcal{M}(h)\ p(h, r) < p(h, r')$ **then**
        $r'' = \arg\min_{r' \in \mathcal{M}(h)} p(h, r')$
        $R = R \cup \{r''\}$
        $\mathcal{M}(h) := \mathcal{M}(h) \backslash \{r''\}$
        $\mathcal{M}(h) := \mathcal{M}(h) \cup \{r\}$
        $R = R \backslash \{r\}$
      end if
    end if
    $i(r)++$
  end for
end while
return $\mathcal{M}$

A stable solution of the Hospital/Residents problem is a mapping of residents to hospitals such that:

- Each hospital accepts maximally $c(h)$ residents;
- No resident r is assigned to a hospital h such that a hospital h' which had a higher preference in $p(r)$ would be willing to admit r and vice-versa.

Note that we assume that there are no ties, i.e., that the functions μ and γ are injective. Given these premises, (Gale and Shapley, 1962) shows that a stable matching always exists. Consequently, Algorithm 11.1 is guaranteed to return a solution. Note that we set the capacity of the hospitals to the smallest whole number that ensures that each resident finds a hospital. Also note that $p(h, r) < p(h, r')$ means that h prefers r over r'.

More details on stable matching can be found in (Manlove et al., 2002).

11.4 THE RAVEN APPROACH

Our approach, dubbed RAVEN (RApid actiVE liNking), addresses the task of linking two knowledge bases S and T by using the active learning paradigm within the pool-based sampling setting (Settles, 2009). Overall, the goal of RAVEN is to find the best classifier $\mathcal{C}$ that achieves the highest possible precision, recall or F_1 score as desired by the user. The algorithm also aims to minimize the burden on the user by limiting the number of link candidates that must be labelled by the user to a minimum.

Algorithm 11.2 The RApid actiVE liNking (RAVEN) algorithm

Require: Source knowledge base $\mathcal{K}_S$
Require: Target knowledge base $\mathcal{K}_T$
1: Find stable class matching between classes of $\mathcal{K}_S$ and $\mathcal{K}_T$
2: Find stable property matching for the selected classes
3: Compute sets S and T; Create initial classifier $\mathcal{C}^0$; $t := 0$
4: **while** termination condition not satisfied **do**
5: Ask the user to classify 2α examples; Update $\mathcal{C}^t$ to $\mathcal{C}^{t+1}$; t := t+1
6: **end while**
7: Compute set $\mathcal{M}$ of links between S and T based on $\mathcal{C}^t$
8: **return** $\mathcal{M}$

An overview of our approach is given in Algorithm 11.2. In a first step, RAVEN aims to detect the restrictions that will define the sets S and T. To achieve this goal, it tries to find a stable matching of pairs of classes, whose instances are to be linked. This is done by applying a two-layered approach and generating a sorted list of class mappings that are presented to the user, who can choose the pair that is to be matched. The second step of our approach consists of finding a stable matching between the properties that describe the instances of the classes specified in the first step. The user is also allowed to alter the suggested matching at will. Based on the property mapping, we compute S and T and generate an initial classifier $\mathcal{C}$ = $\mathcal{C}^0$ in the third step. We then refine $\mathcal{C}$ iteratively by asking the user to classify pairs of instances that are most informative for our algorithm. $\mathcal{C}$ is updated until a termination condition is reached, for example $\mathcal{C}^t$ = $\mathcal{C}^{t+1}$. The final classifier is used to compute the links between S and T, which are returned by RAVEN. In the following, we expand upon each of these three steps.

11.4.1 *Stable Matching of Classes*

The first component of a link specification is a set of restrictions that must be fulfilled by the instances that are to be matched. We present herein how such a matching can be carried out for restrictions that are of the form $\mathcal{R}(x) \leftrightarrow x$ `rdf:type someClass`, as they are the most commonly used restriction. We use a two-layered approach for matching classes in knowledge bases. Our *default approach* begins by selecting a user-specified number of `sameAs` links between the source and the target knowledge base randomly. Then, it computes μ and γ on the classes C_S of $\mathcal{K}_S$ and C_T of $\mathcal{K}_T$ as follows[4]:

$$\mu(C_S, C_T) = \gamma(C_S, C_T) = |\{s \texttt{ type } C_s \wedge s \texttt{ sameAs } t \wedge t \texttt{ type } C_T\}|. \quad (11.4)$$

4 Note that we used `type` to denote `rdf:type` and `owl:sameAs` to denote `sameAs`.

Although several million sameAs links exist on the Web of Data, some knowledge bases that refer to the same entities do not share any links. Consequently, our default approach fails when trying to process such pairs of knowledge bases. In this case, we run our *fallback* approach. It computes μ and γ on the classes of S and T as follows:

$$\mu(C_S, C_T) = \gamma(C_S, C_T) = |\{s \texttt{ type } C_s \wedge s \texttt{ p } x \wedge t \texttt{ type } C_T \wedge t \texttt{ q } x\}|, \tag{11.5}$$

where p and q can be any property.

Let $c(S)$ be the number of classes C_S of S such that $\mu(C_S, C_T) > 0$ for any C_T. Furthermore, let $c(T)$ be the number of classes C_T of T such that $\gamma(C_S, C_T) > 0$ for any C_S. The capacity of each C_T is set to $\lceil c(S)/c(T) \rceil$, thus ensuring that the hospitals provide enough capacity to map all the possible residents. Once μ, γ and the capacity of each hospital has been set, we solve the equivalent $\mathcal{HR}$ problem. It is important to note that although the functions μ and γ are equivalent in both our default and our fallback approaches, the resulting problem is not symmetric, i.e., $p(C_S, C_T) = \zeta$ does not imply that $p(C_T, C_S) = \zeta$.

After the $\mathcal{HR}$ problem has been solved, we present the user with a stable matching sorted in descending order relatively to $\mu(C_S, C_T)$, thereby selecting the pair with the highest $\mu(C_S, C_T)$ as default match. Note that the fallback approach is approximately 30% slower than our default approach. Also, if the fallback approach fails, then we require from the user to enter the class mapping manually.

11.4.2 *Stable Matching of Properties*

The detection of the best matching pairs of properties is very similar to the computation of the best matching classes. For datatype properties p and q, we set:

$$\mu(p, q) = \gamma(p, q) = \{s \texttt{ type } C_s \wedge s \texttt{ p } x \wedge t \texttt{ type } C_T \wedge t \texttt{ q } x\}. \tag{11.6}$$

The initial mapping of properties defines the similarity space in which the link discovery task will be carried out. Note that none of the prior approaches to active learning for record linkage or link discovery automatized this step. We associate each of the basis vectors σ_i of the similarity space to exactly one of the pairs (p, q) of mapping properties detected by RAVEN. Once the restrictions and the property mapping have been specified, we can fetch the elements of the sets S and T.

11.4.3 *Initial Classifier*

The specific formula for the initial linear weighted classifier $\mathcal{L}^0$ results from the formal model presented in Section 11.3 and is given by

$$\mathcal{F}_{\mathcal{L}^0}(s, t) = \sum_{i=1}^{n} \omega_i^0 \sigma_i(s, t). \tag{11.7}$$

Several initialization methods can be used for ω_i^0 and the initial threshold τ^0 of $\mathcal{P}_{\mathcal{L}}$. Here we chose the use the simplest possible approach by setting $\omega_i^0 := 1$ and $\tau^0 := \kappa n$, where $\kappa \in [0, 1]$ is a user-specified *threshold factor*. Note that setting the overall threshold to κn is equivalent to stating that the arithmetic mean of the $\sigma_i(s, t)$ must be equal to κ.

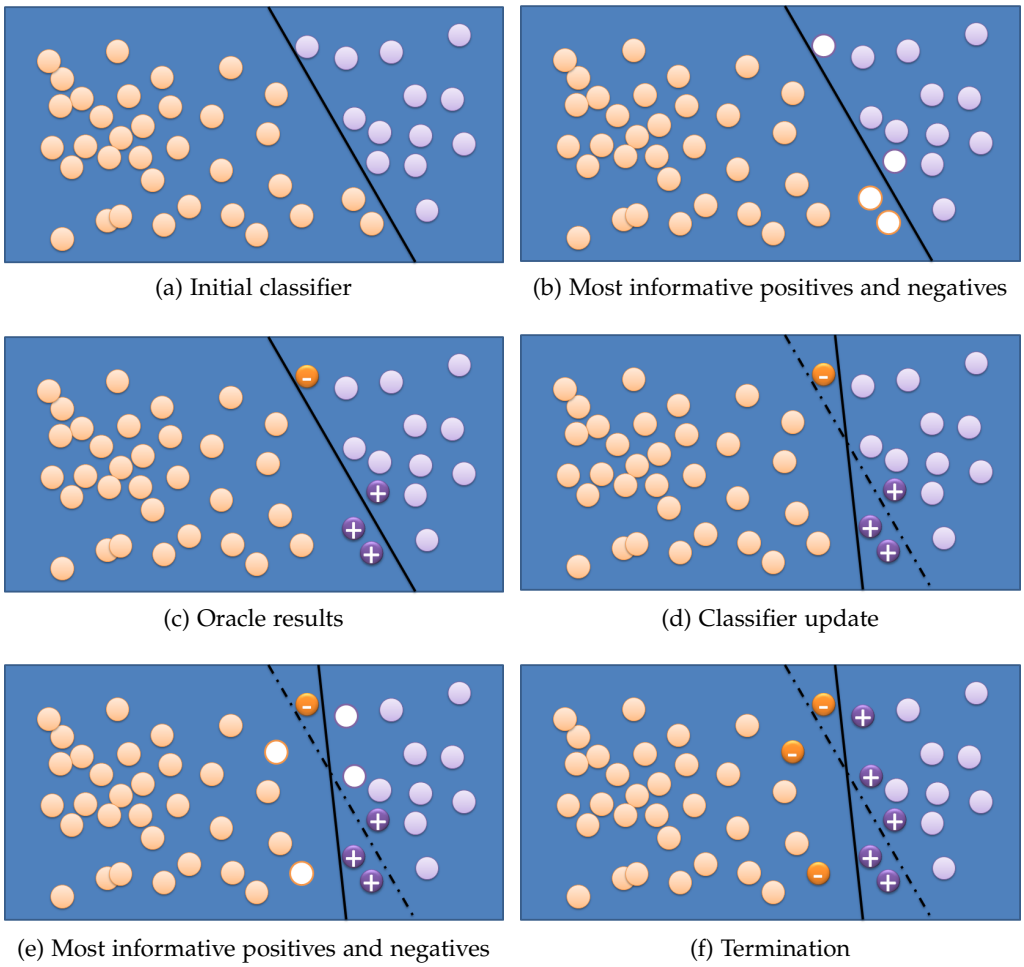
The equivalent initial boolean classifier $\mathcal{B}^0$ is given by

$$\mathcal{F}_{\mathcal{B}}{}^0(s,t) = \bigwedge_{i=0}^{n} (\sigma_i^0(s,t) \geqslant \tau_i^0) \text{ where } \tau_i^0 := \kappa. \tag{11.8}$$

11.4.4 *Updating Classifiers*

RAVEN follows an iterative update strategy, which consists of asking the user to classify 2α elements of $S \times T$ (α is explained below) in each iteration step t and using these to update the values of ω_i^{t-1} and τ_i^{t-1} computed at step $t-1$. The main requirements to the update approach is that it computes those elements of $S \times T$ whose classification allow to maximize the convergence of $\mathcal{C}^t$ to a good classifier and therewith to minimize the burden on the user. The update strategy of RAVEN varies slightly depending on the family of classifiers. In the following, we present how RAVEN updates linear and boolean classifiers.

11.4.4.1 *Updating linear classifiers.*



(a) Initial classifier (b) Most informative positives and negatives

(c) Oracle results (d) Classifier update

(e) Most informative positives and negatives (f) Termination

Figure 11.1: Active learning as implemented by RAVEN

The basic intuition behind our update approach is that given an initial classified $\mathcal{L}^0$, we aim to iteratively present the user with those elements from $S \times T$ whose classification is most unsure until we reach a certain termination condition. An example of such an initial classifier is shown in Figure 11.1a. We call the elements presented to the user *examples*. We call an example *positive* when it is assumed by the classifier to belong to $+$. Else we call it *negative*. In Figure 11.1, the elements that the classifier assigns to the class $+$ are drawn in violet, while the elements of $-$ are drawn in orange. Once the user has provided us with the correct classification for the examples presented to him, the classifier can be updated effectively so as to better approximate the target classifier. In the following, we will define the notion of *most informative example* for linear classifiers before presenting our update approach.

When picturing a classifier as a boundary in the similarity space $\mathfrak{S}$ that separates the classes $+$ and $-$, the examples whose classification is most uncertain are obviously those elements from $S \times T$ who are closest to the boundary specified by the classifier at hand. Note that we must exclude examples that have been classified previously, as presenting them to the user would not improve the classification accuracy of Raven while generating extra burden on the user, who would have to classify the same link candidate twice. Figure 11.1b depicts the idea behind most informative examples for linear classifiers. The elements of $+$ and $-$ that were not previously classified and that are closest to the boundary of $\mathcal{L}$ are selected as being most informative. These are the elements that are presented to the oracle (i.e., the user) for classification. An example of a oracle-given classification is given in Figure 11.1c. The nodes marked with a $+$ where marked by the user as being correct links, while those marked with a $-$ were labelled as incorrect. Here, our classifier only classified one example correctly. Given this information, we update the classifier by using an approach derived from perceptron learning as shown in Figure 11.1d. This classifier update strategy is rather conservative and is based on the assumption that the previous classifier was not completely random and should not be altered too drastically. In our example, using another update approach such as computing a Support Vector Machine based on the user-given classification would have led to a new classifier that would have been almost orthogonal to the initial one. We then iterate the computation of the most informative positive and negative examples (see Figure 11.1e) until a termination condition is reached, e.g., until the classifier can predict the user classification correctly (see Figure 11.1e).

Formally, let $\mathcal{M}^t$ be the set of $(s,t) \in S \times T$ classified by $\mathcal{L}^t$ as belonging to $+1$. Furthermore, let $\mathcal{P}^{t-1}$ (resp. $\mathcal{N}^{t-1}$) be the set of examples that have already been classified by the user as being positive examples, i.e, links (resp. negative examples, i.e., wrong links) in the first $t-1$ iterations. We define a set Λ as being a set of most informative examples λ for $\mathcal{L}^{t+1}$ when the following two conditions hold:

$$\forall \lambda \in S \times T \; (\lambda \in \Lambda \rightarrow \lambda \notin \mathcal{P}^{t-1} \cup \mathcal{N}^{t-1}) \tag{11.9}$$

$$\forall \lambda' \notin \mathcal{P}^{t-1} \cup \mathcal{N}^{t-1} : \lambda' \neq \lambda \rightarrow |{\mathcal{F}_{\mathcal{L}}}^t(\lambda') - \tau^t| \geqslant |{\mathcal{F}_{\mathcal{L}}}^t(\lambda) - \tau^t|. \tag{11.10}$$

Note that there are several sets of most informative examples of a given magnitude. We denote a set of most informative examples of magnitude α by Λ_α. A set of most informative positive examples, Λ^+, is a set of pairs such that

$$\forall \lambda \notin \Lambda^+ \cup \mathcal{P}^{t-1} \cup \mathcal{N}^{t-1} : ({\mathcal{F}_{\mathcal{L}}}^t(\lambda) < \tau^t) \vee (\forall \lambda^+ \in \Lambda^+ : {\mathcal{F}_{\mathcal{L}}}^t(\lambda) > {\mathcal{F}_{\mathcal{L}}}^t(\lambda^+)). \tag{11.11}$$

In words, Λ^+ is the set of examples that belong to class $+$ according to $\mathcal{C}$ and are closest to $\mathcal{C}$'s boundary. Similarly, the set of most informative negative examples, Λ^-, is the set of examples such that

$$\forall \lambda \notin \Lambda^- \cup \mathcal{P}^{t-1} \cup \mathcal{N}^{t-1} : (\mathcal{F}_{\mathcal{L}}{}^t(\lambda) \geqslant \tau^t) \vee (\forall \lambda^- \in \Lambda^- : \mathcal{F}_{\mathcal{L}}{}^t(\lambda) < \mathcal{F}_{\mathcal{L}}{}^t(\lambda^-)). \quad (11.12)$$

We denote a set of most informative (resp. negative) examples of magnitude α as Λ^+_α (resp. Λ^-_α). The 2α examples presented to the user consist of the union $\Lambda^+_\alpha \cup \Lambda^-_\alpha$, where Λ^+_α and Λ^-_α are chosen randomly amongst the possible sets of most informative positive resp. negative examples .

The update rule for the weights of $\mathcal{L}^t$ is derived from the well-known Perceptron algorithm, i.e.,

$$\omega_i^{t+1} = \omega_i^t + \eta^+ \sum_{\lambda \in \Lambda^+} \rho(\lambda)\sigma_i(\lambda) - \eta^- \sum_{\lambda \in \Lambda^-} \rho(\lambda)\sigma_i(\lambda), \quad (11.13)$$

where η^+ is the learning rate for positives examples, η^- is the learning rate for negative examples and $\rho(\lambda)$ is 0 when the classification of λ by the user and $\mathcal{L}^t$ are the same and 1 when they differ.

The threshold is updated similarly, i.e,

$$\tau_i^{t+1} = \tau_i^t + \eta^+ \sum_{\lambda \in \Lambda^+_\alpha} \rho(\lambda)\mathcal{F}_{\mathcal{L}}{}^t(\lambda) - \eta^- \sum_{\lambda \in \Lambda^-_\alpha} \rho(\lambda)\mathcal{F}_{\mathcal{L}}{}^t(\lambda). \quad (11.14)$$

Note that the weights are updated by using the dimension which they describe while the threshold is updated by using the whole specific function. Finally, the sets $\mathcal{P}^{t-1}$ and $\mathcal{N}^{t-1}$ are updated to

$$\mathcal{P}^t := \mathcal{P}^{t-1} \cup \Lambda^+_\alpha \quad (11.15)$$

and

$$\mathcal{N}^t := \mathcal{N}^{t-1} \cup \Lambda^-_\alpha. \quad (11.16)$$

11.4.4.2 *Updating boolean classifiers*

The notion of *most informative example* differs slightly for boolean classifiers. λ is considered a most informative example for $\mathcal{B}$ when the conditions

$$\lambda \notin \mathcal{P}^{t-1} \cup \mathcal{N}^{t-1} \quad (11.17)$$

and

$$\forall \lambda' \notin \mathcal{P}^{t-1} \cup \mathcal{N}^{t-1} : \lambda' \neq \lambda \rightarrow \sum_{i=1}^{n} |\sigma_i^t(\lambda') - \tau_i^t| \geqslant \sum_{i=1}^{n} |\sigma_i^t(\lambda) - \tau_i^t| \quad (11.18)$$

hold. The update rule for the thresholds τ_i^t of $\mathcal{B}$ is then given by

$$\tau_i^{t+1} = \tau_i^t + \eta^+ \sum_{\lambda \in \Lambda^+_\alpha} \rho(\lambda)\sigma_i(\lambda) - \eta^- \sum_{\lambda \in \Lambda^-_\alpha} \rho(\lambda)\sigma_i(\lambda), \quad (11.19)$$

where η^+ is the learning rate for positives examples, η^- is the learning rate for negative examples and $\rho(\lambda)$ is 0 when the classification of λ by the user and $\mathcal{C}_{t-1}$ are the same and 1 when they differ. The sets $\mathcal{P}^{t-1}$ and $\mathcal{N}^{t-1}$ are updated as given in Equation 11.15 and Equation 11.16.

11.5 EXPERIMENTS AND RESULTS

11.5.1 *Experimental Setup*

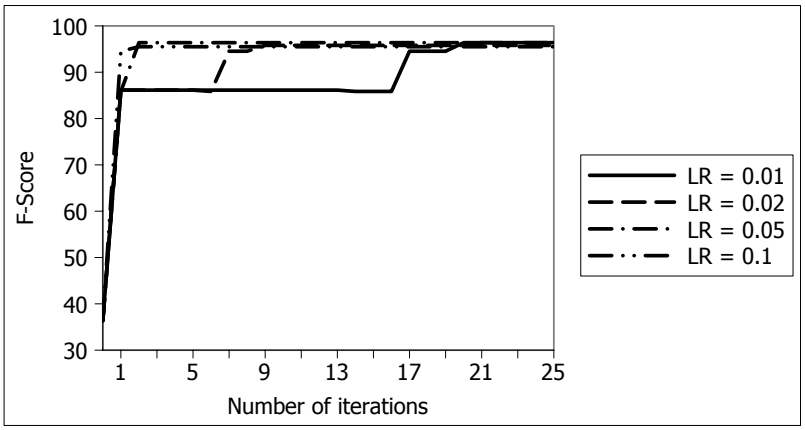
We carried out three series of experiments to evaluate our approach. In our first experiment, dubbed *Diseases*, we aimed to map diseases from DBpedia with diseases from Diseasome. In the *Drugs* experiments, we linked drugs from Sider with drugs from Drugbank. Finally, in the *Side-Effects* experiments, we aimed to link side-effects of drugs and diseases in Sider and Diseasome.

In all experiments, we aimed to compute how well linear and boolean classifiers learned by RAVEN could approximate a manually specified configuration for mapping two knowledge bases. We used the following setup: The same learning rates η^+ and η^- were set to the same value η, which we varied between 0.01 and 0.1. We set the number of inquiries per iteration to 4. The threshold factor κ was set to 0.8. In addition, the number of instances used during the automatic detection of class resp. property matches was set to 100 resp. 500. If no class mapping was found via `sameAs` links, then the fallback solution was called and compared the property values of 1000 instances chosen randomly from the source and target knowledge bases. We used the trigrams metric as default similarity measure for strings and the Euclidean similarity as default similarity measure for numeric values. To measure the quality of our results, we used the well-know precision, recall and F-score measures. We also measured the total number of inquiries that RAVEN needed to reach its maximal F-Score. As reference data, we used the set of instances that mapped perfectly according to a configuration created manually. All experiments were carried out on a computer running Windows 7 Professional SP1 (32-bit) Build 7601 on an Intel Core2 Duo with 2.53GHz and 3072MB RAM.

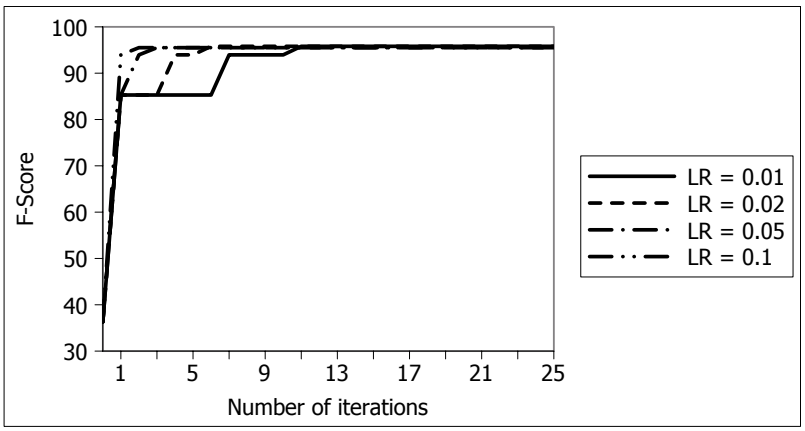
11.5.2 *Results*

Table 11.1 presents an excerpt of the mappings computed automatically by RAVEN. The elements of the stable matching computed from these mapping were used as initial configuration. The results of our experiments are shown in Figure 11.2, Figure 11.3 and Figure 11.4. The first experiment, Diseases, proved to be the most difficult for RAVEN. Although the `sameAs` links between `Diseasome` and `DBpedia` allowed our experiment to run without making use of the fallback solution, we had to send 12 inquiries to the user when the learning rate was set to 0.1 to determine the best configuration that could be learned by linear and boolean classifiers. Smaller learning rates led to the system having to send even up to 80 inquiries ($\eta = 0.01$) to determine the best configuration. In this experiment linear classifiers outperform boolean classifiers in all setups by up to 0.8% F-score.

The second and the third experiment display the effectiveness of RAVEN. Although the fallback solution had to be used in both cases, our approach is able to detect the right configuration with an accuracy of even 100% in the Side-Effects experiment by asking the user no more than 4 questions. This is due to the linking configuration of the user leading to two well-separated sets of instances. In these cases, RAVEN converges rapidly and finds a good classifier rapidly. Note that in these two cases, all learning rates in combination with both linear and boolean classifiers led to the same results (see Figure 11.3 and Figure 11.4).



(a) Linear classifier



(b) Boolean classifier

Figure 11.2: Learning curves on Diseases experiments

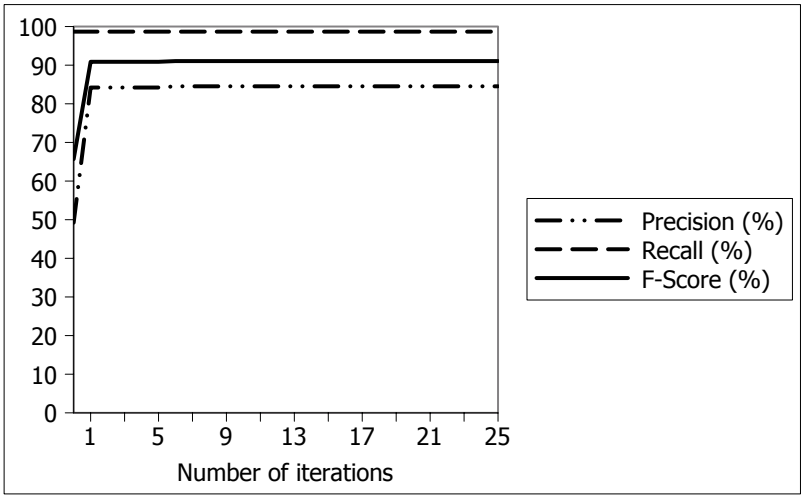


Figure 11.3: Learning curve of the Drugs experiments

Although we cannot directly compare our results to other approaches as it is the first active learning algorithm for learning link specifications, results reported in the database area suggest that RAVEN achieves state-of-the-art performance. The runtimes required for each iteration ensure that our approach can be used in real-

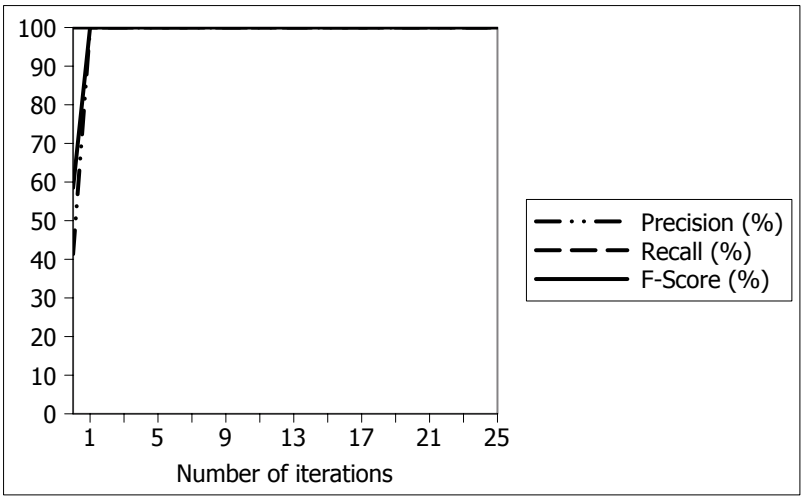


Figure 11.4: Learning curve of the Side-Effect experiment

world interactive scenarios. In the worst case, the user has to wait for 1.4 seconds between two iterations as shown in Figure 11.5. The runtime for the computation of the initial configuration depends heavily on the connectivity to the SPARQL endpoints. In our experiments, the computation of the initial configuration demanded 60 seconds when `sameAs` links existed between the knowledge bases. When the fallback solution was used, the runtimes increased and reached 90 seconds.

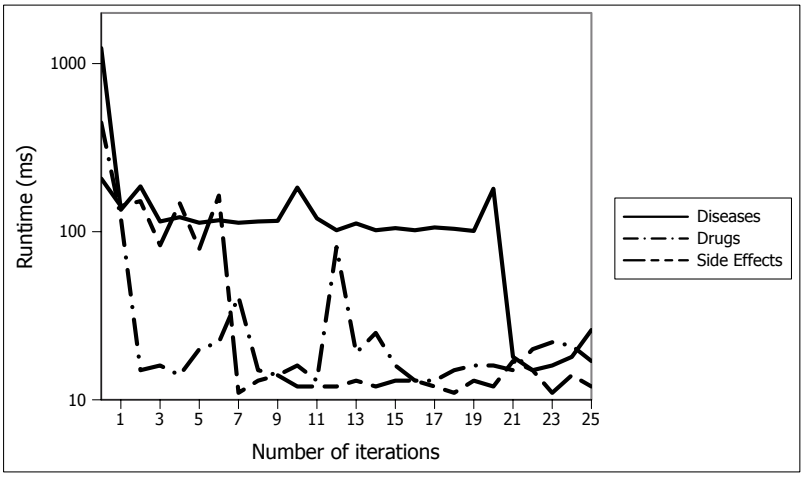


Figure 11.5: Average runtimes for each iteration

11.6 CONCLUSION AND FUTURE WORK

In this paper, we presented RAVEN, the first active learning approach tailored towards semi-automatic Link Discovery on the Web of Data. We showed how RAVEN uses stable matching algorithms to detect initial link configurations. We opted to use the solution of the hospital residence problem ($\mathcal{HR}$) without ties because of the higher time complexity of the solution of $\mathcal{HR}$ with ties, i.e., L^4, where L is the size of the longest preference list, i.e., $\max(|R|, |H|)$. Still, our work could be extended by measuring the effect of considering ties on the matching computed by RAVEN. Our experimental results showed that RAVEN can compute accurate link specifica-

tions (F-score between 90% and 100%) by asking the user to classify a very small number of positive and negative examples (between 4 and 12 for a learning rate of 0.1). Our results also showed that our approach can be used in an interactive scenario because of LIMES' time efficiency, which allowed to compute new links in less than 1.5 seconds in the evaluation tasks. The advantages of this interactive approach can increase the quality of generated links while reducing the effort to create them. The RAVEN algorithm as well as a graphical user interface will be made available as open source within the SAIM[5] framework.

In future work, we will explore how to detect optimal values for the threshold factor κ automatically, for example, by using clustering approaches. In addition, we will investigate the automatic detection of domain-specific metrics that can model the idiosyncrasies of the dataset at hand. Another promising extension to RAVEN is the automatic detection of the target knowledge base to even further simplify the linking process, since users often might not even be aware of appropriate linking targets (see (Guéret et al., 2010) for research in this area). By these means, we aim to provide the first zero-configuration approach to Link Discovery.

5 http://saim.sf.net

Experiment	Class Mapping	Property Mapping
Diseases	ds:diseases→dbp:Disease*	rdfs:label → dbp:name*
	ds:diseases→yago:Disease114070360	rdfs:label → foaf:name
	ds:diseases→yago:NeurologicalDisorders	rdfs:label → rdfs:label
	ds:diseases→yago:TypesOfCancer	ds:name → dbp:name
	ds:diseases→yago:GeneticDisorders	ds:name → foaf:name*
	ds:diseases→yago:BloodDisorders	ds:name → rdfs:label
	ds:diseases→yago:TransmissibleSpongiformEncephalopathies	
	ds:diseases→yago:Syndromes	
	ds:diseases→yago:ChromosomeInstabilitySyndromes	
	ds:diseases→yago:PigmentDisorders	
	ds:diseases→yago:CongenitalDisorders	
Drugs	dbk:drugs→sd:drugs*	dbk:brandName → sd:drugName*
	dbk:drugs→sd:Offer	dbk:genericName → sd:drugName
		rdfs:label → sd:drugName
		dbk:brandName → rdfs:label
Side-Effects	sd:sideEffects→ds:diseases*	rdfs:label → rdfs:label*
		rdfs:label → ds:name
		sd:sideEffectName → rdfs:label
		sd:sideEffectName → ds:name*

Table 11.1: Initial property and class mappings computed by RAVEN in our experiments. The class and property mappings marked with an asterix were returned by the stable matching algorithm and used in the classifier.

12

EAGLE: LEARNING LINK SPECIFICATIONS USING GENETIC PROGRAMMING

This chapter presents EAGLE, an active learning approach based on genetic programming. EAGLE generates highly accurate link specifications while reducing the annotation burden for the user. The content of the chapter is taken from (Ngonga Ngomo and Lyko, 2012). The author designed the approach, co-implemented the solution, designed the implementation and co-wrote the paper.

12.1 INTRODUCTION

The growth of the Linked Data Web over the last years has led to a compendium of currently more than 30 billion triples (Auer et al., 2013a). Yet, it still contains a relatively low number of links between knowledge bases (less than 2% at the moment). Devising approaches that address this problem still remains a very demanding task. This is mainly due to the difficulty behind Link Discovery being twofold: First, the quadratic complexity of Link Discovery requires time-efficient approaches that can efficiently compute links when given a specification of the conditions under which a link is to be built (Isele et al., 2011; Ngonga Ngomo and Auer, 2011) (i.e., when given a so-called *link specification*). Such specifications can be of arbitrary complexity, ranging from a simple comparison of labels (e.g., for finding links between countries) to the comparison of a large set of features of different types (e.g., using population, elevation and labels to link villages across the globe). In previous work, we have addressed this task by developing the LIMES[1] framework. LIMES provides time-efficient approaches for Link Discovery and has been shown to outperform other frameworks significantly (Ngonga Ngomo, 2011).

The second difficulty behind Link Discovery lies in the detection of accurate link specifications. Most state-of-the-art Link Discovery frameworks such as LIMES and SILK (Isele et al., 2011) adopt a property-based computation of links between entities. To ensure that links can be computed with a high accuracy, these frameworks provide (a) a large number of similarity measures (i.e., Levenshtein, Jaccard for strings) for comparing property values and (b) manifold means for combining the results of these measures to an overall similarity value for a given pair of entities. When faced with this overwhelming space of possible combinations, users often adapt a time-demanding trial-and-error approach to detect an accurate link specification for the task at hand. There is consequently a blatant need for approaches that support the user in the endeavor of finding accurate link specifications. From a user's perspective, approaches for the semi-automatic generation of link specification must support the user by

1. reducing the time frame needed to detect a link specification (time efficiency),
2. generating link specifications that generate a small number of false positives and negatives (accuracy) and

1 http://limes.sf.net

3. providing the user with easily readable and modifiable specifications (readability).

In this paper, we present the EAGLE algorithm, a supervised machine-learning algorithm for the detection of link specifications that abides by the three criteria presented above. One of the main drawbacks of machine-learning approaches is that they usually require a large amount of training data to achieve a high accuracy. Yet, the generation of training data can be a very tedious process. EAGLE surmounts this problem by implementing an active learning approach (Settles, 2009). Active learning allows the interactive annotation of highly informative training data. Therewith, active learning approaches can minimize the amount of training data necessary to compute highly accurate link specifications.

Overall, the contributions of this paper are as follows:

- We present a novel active learning approach to learning link specifications based on genetic programming.
- We evaluate our approach on three different datasets and show that we reach F-measures of above 90% by asking between 10 and 20 questions even on difficult datasets.
- We compare our approach with state-of-the-art approaches on the DBLP-ACM dataset and show that we outperform them with respect to runtime while reaching a comparable accuracy.

The advantages of our approach are manifold. In addition to its high accuracy, it generates readable link specifications which can be altered by the user at will. Furthermore, given the superior runtime of LIMES on string and numeric properties, our approach fulfills the requirements for use in an interactive setting. Finally, our approach only requires very small human effort to discover link specifications of high accuracy as shown by our evaluation.

The rest of this paper is organized as follows: First, we give a brief overview of the state of the art. Thereafter, we present the formal framework within which EAGLE is defined. This framework is the basis for the subsequent specification of our approach. We then evaluate our approach with several parameters on three different datasets. We demonstrate the accuracy of our approach by computing its F-measure. Moreover, we show that EAGLE is time-efficient by comparing its runtime with that of other approaches on the ACM-DBLP dataset. We also compare our approach with its non-active counterpart and study when the use of active learning leads to better results.

12.2 RELATED WORK

Over the last years, several approaches have been developed to address the time complexity of link discovery. Some of these approaches focus on particular domains of applications. For example, the approach implemented in RKB knowledge base (RKB-CRS) (Glaser et al., 2009) focuses on computing links between universities and conferences while GNAT (Raimond et al., 2008) discovers links between music datasets. Further simple or domain-specific approaches can be found in (Cudré-Mauroux et al., 2009; Hogan et al., 2010; Nikolov et al., 2009; Papadakis et al., 2011; Sleeman and Finin, 2010). In addition, domain-independent

approaches have been developed, that aim to facilitate link discovery all across the Web. For example, RDF-AI (Scharffe et al., 2009) implements a five-step approach that comprises the preprocessing, matching, fusion, interlinking and post-processing of datasets. SILK (Isele et al., 2011) is a time-optimized tool for link discovery. It implements a multi-dimensional blocking approach that is guaranteed to be lossless thanks to the overlapping blocks it generates. Another lossless Link Discovery framework is LIMES (Ngonga Ngomo, 2011), which addresses the scalability problem by implementing time-efficient similarity computation approaches for different data types and combining those using set theory. Note that the task of discovering links between knowledge bases is closely related with record linkage (Bleiholder and Naumann, 2008; Elmagarmid et al., 2007; Köpcke et al., 2009; Winkler, 2006). To the best of our knowledge, the problem of discovering accurate link specifications has only been addressed in very recent literature by a small number of approaches: The SILK framework (Isele et al., 2011) now implements a batch learning approach to discovery link specifications based on genetic programming which is similar to the approach presented in (de Carvalho et al., 2008). The algorithm implemented by SILK also treats link specifications as trees but relies on a large amount of annotated data to discover high-accuracy link specifications. The RAVEN algorithm (Ngonga Ngomo et al., 2011) on the other hand is an active learning approach that treats the discovery of specifications as a classification problem. It discovers link specifications by first finding class and property mappings between knowledge bases automatically. RAVEN then uses these mappings to compute linear and boolean classifiers that can be used as link specifications. A related approach that aims to detect discriminative properties for linking is that presented by (Song and Heflin, 2011). In addition to these approaches, several machine-learning approaches have been developed to learn classifiers for record linkage. For example, machine-learning frameworks such as FEBRL (Christen, 2008) and MARLIN (Bilenko and Mooney, 2003) rely on models such as Support Vector Machines (Cristianini and Ricci, 2008; Keerthi and Lin, 2003), decision trees (Yuan and Shaw, 1995) and rule mining (Agrawal et al., 1993) to detect classifiers for record linkage. Our approach, EAGLE, goes beyond previous work in three main ways. First, it is an active learning approach. Thus, it does not require the large amount of training data required by batch learning approaches such as FEBRL, MARLIN and SILK. Furthermore, it allows to use the full spectrum of operations implemented in LIMES. Thus, it is not limited to linear and boolean classifiers such as those generated by FeBRL and RAVEN. Finally, it can detect property and class mappings automatically. Thus, it does not need to be seeded to converge efficiently like previous approaches (Isele et al., 2011).

12.3 PRELIMINARIES

In the following, we present the core of the formalization and notation necessary to implement EAGLE. We first formalize the Link Discovery problem. Then, we give an overview of the grammar that underlies links specifications in LIMES and show how the resulting specifications can be represented as trees. We show how the discovery of link specifications can consequently be modeled as a genetic programming problem. Subsequently, we give some insight in active learning and then present the active learning model that underlies our work.

12.3.1 *Link Discovery*

The link discovery problem, which is similar to the record linkage problem, is an ill-defined problem and is consequently difficult to model formally (Arasu et al., 2010). In general, link discovery aims to discover pairs $(s,t) \in S \times T$ related via a relation R.

Definition 8 (Link Discovery)**.** *Given two sets* S *(source) and* T *(target) of entities, compute the set* $\mathcal{M}$ *of pairs of instances* $(s,t) \in S \times T$ *such that* $R(s,t)$*.*

The sets S resp. T are usually (not necessarily disjoint) subsets of the instances contained in two (not necessarily disjoint) knowledge bases $\mathcal{K}_S$ resp. $\mathcal{K}_T$. One way to automate this discovery is to compare the $s \in S$ and $t \in T$ based on their properties using a (complex) similarity measure σ. Two entities $s \in S$ and $t \in T$ are then considered to be linked via R if $\sigma(s,t) \geqslant \theta$ (Ngonga Ngomo and Auer, 2011). The specification of the sets S and T and of this similarity condition is usually carried out within a *link specification*.

Definition 9 (Link Specification)**.** *A link specification consists of three parts: (1) two sets of restrictions* $\mathcal{R}_1^S$ *...* $\mathcal{R}_m^S$ *resp.* $\mathcal{R}_1^T$ *...* $\mathcal{R}_k^T$ *that specify the sets* S *resp.* T*, (2) a specification of a complex similarity metric* σ *via the combination of several atomic similarity measures* σ_1*, ...,* σ_n *and (3) a set of thresholds* θ_1*, ...,* θ_n *such that* θ_i *is the threshold for* σ_i*.*

A restriction $\mathcal{R}$ is generally a logical predicate. Typically, restrictions in link specifications state the `rdf:type` of the elements of the set they describe, i.e., $\mathcal{R}(x) \leftrightarrow x$ `rdf:type someClass` or the features that the elements of the set must have, e.g., $\mathcal{R}(x) \leftrightarrow (\exists y : x$ `someProperty` $y)$. Each $s \in S$ must abide by each of the restrictions $\mathcal{R}_1^S$ ... $\mathcal{R}_m^S$, while each $t \in T$ must abide by each of the restrictions $\mathcal{R}_1^T$ ... $\mathcal{R}_k^T$. Each similarity σ_i is used to compare pairs of property values of instances s and t. EAGLE relies on the approach presented in (Ngonga Ngomo, 2011) to detect the class and property mappings. Consequently, in the remainder of this paper, the term *link specification* will be used to denote the complex similarity condition used to determine whether two entities should be linked.

12.3.2 *Link Specifications as Trees*

Our definition of a link specification relies on the definition of *atomic similarity measures* and *similarity measures*. Generally, a similarity measure m is a function such that $m : S \times T \to [0,1]$. We call a measure atomic (dubbed $atomicMeasure$) when it relies on exactly one similarity measure σ (e.g., trigrams similarity for strings) to compute the similarity of a pair $(s,t) \in S \times T$. A similarity measure m is either an atomic similarity measure $atomicMeasure$ or the combination of two similarity measures via a metric operator $metricOp$ such as MAX, MIN and linear combinations ADD.

1. $m \rightsquigarrow atomicMeasure$
2. $m \rightsquigarrow metricOp(m_1, m_2)$

We call a link specification atomic ($atomicSpec$) if it compares the value of a measure m with a threshold θ, thus returning the pairs (s,t) that satisfy the condition $\sigma(s,t) \geqslant \theta$. A link specification $spec(m,\theta)$ is either an atomic link specification or

Figure 12.1: Atomic measure

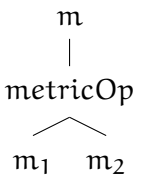


Figure 12.2: Complex measure

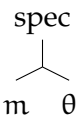


Figure 12.3: Atomic specification

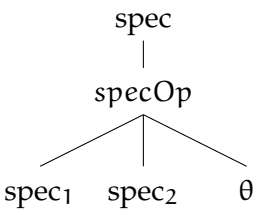


Figure 12.4: Complex specification

the combination of two link specifications via specification operators specOp such as AND (intersection of the set of results of two specs), OR (union of the result sets), XOR (symmetric set difference), or DIFF (set difference). Thus, the following grammar for specifications holds :

1. $spec(m, \theta) \rightsquigarrow atomicSpec(m, \theta)$
2. $spec(m, \theta) \rightsquigarrow specOp(spec(m_1, \theta_1), spec(m_2, \theta_2))$

Each link specification that abides by the grammar specified above can be consistently transformed into a tree that contains the central constructs shown in Figure 12.1, Figure 12.2, Figure 12.3 and Figure 12.4.

12.4 APPROACH

As we have formalized link specifications as trees, we can use Genetic Programming (GP) to solve the problem of finding the most appropriate complex link specification for a given pair of knowledge bases. Given a problem, the basic idea behind genetic programming (Koza, 1992) is to generate increasingly better solutions of the given problem by applying a number of genetic operators to the current population. In the following, we will denote the population at time t by g^t. Genetic operators simulate natural selection mechanisms such as mutation and reproduction to enable the creation of individuals that best abide by a given fitness function. One of the key problems of genetic programming is that it is a non-deterministic procedure. In addition, it usually requires a large training dataset to detect accurate solutions. In this paper, we propose the combination of GP and active learning (Settles, 2009). Our intuition is that by merging these approaches, we can infuse some determinism in the GP procedure by allowing it to select the most informative data for the population. Thus, we can improve the convergence of GP approaches while reducing the labeling effort necessary to use them. In the following, we present our implementation of the different GP operators on link specifications and how we combine GP and active learning.

12.4.1 *Overview*

Algorithm 12.1 gives an overview of the approach implemented by EAGLE. After the detection of matching classes and properties using the approach explicated in (Ngonga Ngomo et al., 2011), we begin by generating a random population of individual link specifications. To evolve a population to the next one a number of steps is required: First, all existing individuals must be assigned a fitness score. This score reflects how well a link specification performs on the training data at hand. Subsequently, the genetic operators reproduction, mutation and crossover are applied to the individuals of the current population in order to let the individuals adapt to the problem. The *reproduction* determines which individual is carried into the next generation. Note that throughout this paper, we use a tournament setting of two randomly chosen individuals. The *mutation* operator can applied to a single individual. It alters this individual by randomly changing parts of its tree (i.e., of his genome) with the aim of creating a new individual. The *crossover* operator also aims at generating new individuals. It achieves this goal by crossing over branches of the program trees of two parent individuals.

Algorithm 12.1 Main EAGLE algorithm

Require: Specification of the two knowledge bases KS and KT
  Get set S and set T of instances as specified in KS respectively KT.
  Get property mapping (KS, KT)
  Get reference mapping by asking user to label n random pairs $(s, t) \in S \times T$
  repeat
    Evolve population(population,size) generations times.
    Compute n most informative link candidates and ask user to label them.
  until stop condition reached

Algorithm 12.2 Evolution of a population

  if population is empty **then**
    create size random individuals
  end if
  Compute fitness of population
  Apply genetic operators to population
  return population

In the following, we will explicate each of the steps of our algorithm in more detail. Each of these steps will be exemplified by using the link specification shown in Figure 12.5.

12.4.2 *Evolution of population*

Evolution is the primary process which enables GP to solve problems and drives the development of efficient solutions for a given problem. At the beginning of our computation the population is empty and must be built by individuals generated randomly. This is carried out by generating random trees whose nodes are filled with functions or terminals as required. For this paper, we defined the operators (functions and terminals) in the genotype for the problem to generate link speci-

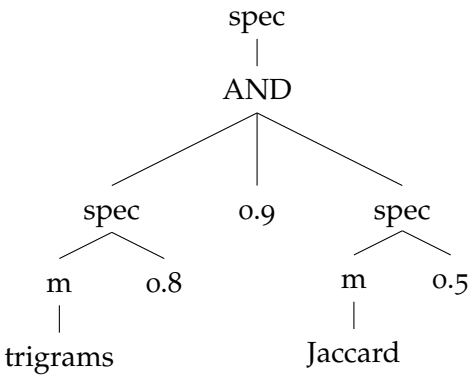


Figure 12.5: Examplary link specification

fications as follows: all metricOp and specOp were set to be functions. Terminal symbols were thresholds and measures. Note that these operators can be extended at will. In addition, all operators were mapped to certain constraints so as to ensure that EAGLE only generates valid program trees. For example, the operator that compares numeric properties only accepts terminals representing numeric properties from the knowledge bases.

Let g^t be the population at the iteration t. To evolve a population to the generation g^{t+1} we first determine the fitness of all individuals of generation g^t (see Section 12.4.3). These fitness values build the basis for selecting individuals for the genetic operator reproduction. We use a tournament setting between two selected individuals to decide which one is copied to the next generation g^{t+1}. On randomly selected individuals the operator mutation is applied according with a probability called the *mutation rate*. The mutation operator changes single nodes in the program tree of the individuals. A mutation can affect an individual in three different ways: First, it can alter the thresholds used by the individual. Second, a mutation can alter the properties contained in the individual's genome. Finally, mutations can modify the measures included in the individuals (see Figure 12.6). The third genetic operator, *crossover*, operates on two parent individuals and builds a new offspring by swapping two random subtrees of the parent genotypes. Figure 12.7 exemplifies the functionality of the crossover operator.

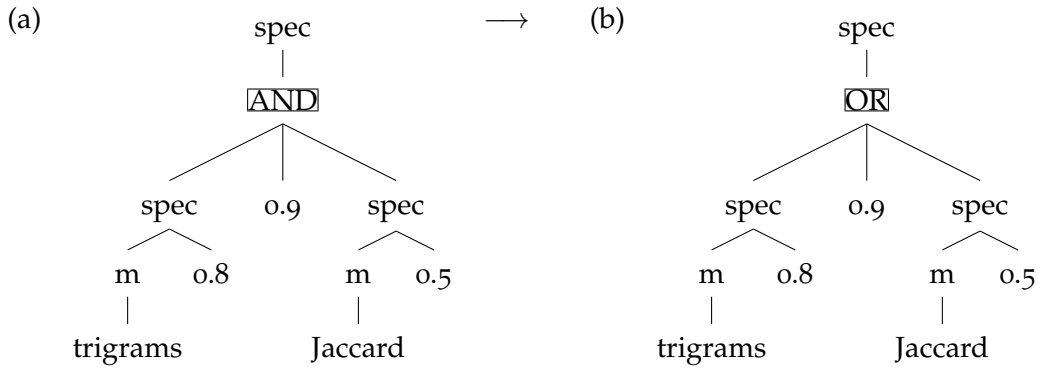


Figure 12.6: Mutation example. Mutation changes boolean operator.

The individuals selected to build the population of g^{t+1} are the n fittest from the union of the set of newly created individuals and g^t. Note that we iteratively generate new populations of potential fitter individuals.

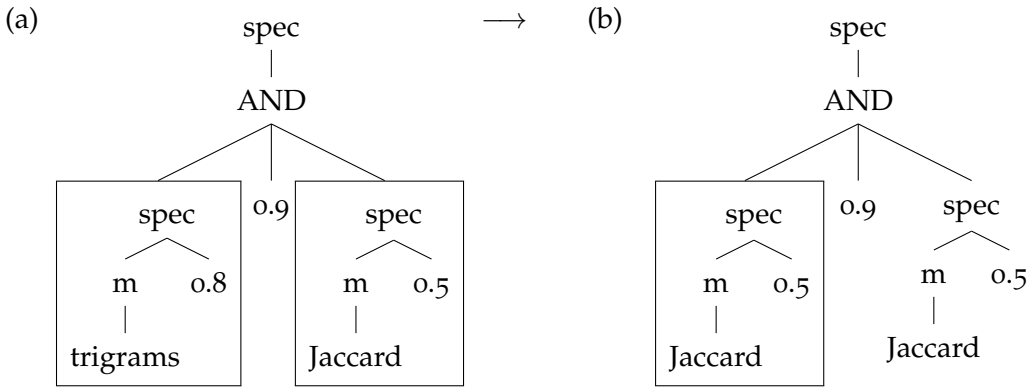


Figure 12.7: Crossover example. Consider we have two individuals with a program tree like in (a). A crossover operation can replace subtrees to produce an offspring like (b).

12.4.3 *Fitness*

The aim of the fitness function is to approximate how well a solution (i.e., a link specification) solves the problem at hand. In the supervised machine learning setting, this is equivalent to computing how well a link specification maps the training data at hand. To determine the fitness of an individual we first build the link specification that is described by the tree at hand. Given the set of available training data $\mathcal{O} = \{(x_i, y_i) \in S \times T\}$, we then run the specification by using the sets $S(\mathcal{O}) = \{s \in S : \exists t \in T : (s,t) \in \mathcal{O}\}$ and $T(\mathcal{O}) = \{t \in T : \exists s \in S : (s,t) \in \mathcal{O}\}$. The result of this process is a mapping $\mathcal{M}$ that is then evaluated against $\mathcal{O}$ by the means of the standard F-measure defined as

$$\frac{2PR}{P+R} \text{ where } P = \frac{|\mathcal{M} \cap \mathcal{O}|}{|\mathcal{M}|} \text{ and } R = \frac{|\mathcal{M} \cap \mathcal{O}|}{|\mathcal{O}|}. \tag{12.1}$$

Note that by running the linking on $S(\mathcal{O})$ and $T(\mathcal{O})$, we can significantly reduce EAGLE's runtime.

12.4.4 *Computation of most informative link candidates*

The main idea behind the reduced of the amount of labeling effort required by active learning approaches is that they only required highly informative training data from the user. Finding these most informative pieces of information is usually carried out by measuring the amount of information that the labeling of a training data item would bear. Given the setting of EAGLE in which several possible solutions co-exist, we opted for applying the idea of active learning by committees as explicated in (Liere and Tadepalli, 1997). The idea here is to consistently entertain a finite and incomplete set of solutions to the problem at hand. The most informative link candidates are then considered to be the pairs $(s,t) \in S \times T$ upon which the different solutions disagree the most. In our case, these are the link candidates that maximize the disagreement function $\delta((s,t))$:

$$\delta((s,t)) = (n - |\{\mathcal{M}_i^t : (s,t) \in \mathcal{M}_i\}|)(n - |\{\mathcal{M}_i^t : (s,t) \notin \mathcal{M}_i\}|), \tag{12.2}$$

where $\mathcal{M}_i$ are the mappings generated by the population g^t. The pairs (s,t) that lead to the highest disagreement score are presented to the user, who provides

Label	S	T	$\|S\| \times \|T\|$	Oracle size
Drugs	Dailymed	Drugbank	1.09×10^6	1046
Movies	DBpedia	LinkedMDB	1.12×10^6	1056
Publications	ACM	DBLP	6.01×10^6	2224

Table 12.1: Characteristics of the datasets used for the evaluation of EAGLE. S stands for source, T for target.

the system with the correct labels. This training set is finally updated and used to compute the next generations of solutions.

12.5 EVALUATION

12.5.1 *Experimental Setup*

We evaluated our approach in three experiments. In our experiments, our main goal not only to show that we can discover link specifications of different complexity with high accuracy. In addition, we also aimed to study the effect of the population size and of active learning on the quality of link specifications. For this purpose, we devised three experiments whose characteristics are shown in Table 12.1.

The goal of the first experiment, called Drugs, was to measure how well we can detect a manually created LIMES specification. For this purpose, we generated `owl:sameAs` link candidates between Drugs in DailyMed and Drugbank by using their `rdfs:label`. The second experiment, Movies, was carried out by using the results of a LATC[2] link specification. Here, we fetched the links generated by a link specification that linked movies in DBpedia to movies in LinkedMDB (Hassanzadeh and Consens, 2009), gathered the `rdfs:label` of the movies as well as the `rdfs:label` of their directors in the source and target knowledge bases and computed a specification that aimed to reproduce the set of links at hand as exactly as possible. Note that this specification is hard to reproduce as the experts who created this link specification applied several transformations to the property values before carrying out the similarity computation that led to the results at hand. Finally, in our third experiment (Publications), we used the ACM-DBLP dataset described in (Köpcke et al., 2009). Our aim here was to compare our approach with other approaches with respect to both runtime and F-measure.

All experiments were carried out on one kernel of an AMD Opteron Quad-Core processor (2GHz) with the followings settings: the population size was set to 20 or 100. The maximal number of generations was set to 50. In all active learning experiments, we carried out 10 inquiries per iteration cycle. In addition, we had the population evolve for 10 generations between all inquiries. The mutation and crossover rates were set to 0.6. For the batch learners, we set the number of generations to the size of the training data. Note that this setup is of disadvantage for active learning as the batch learners then have more data and more iterations on the data to learn the best possible specification. We used this setting as complementary for the questions that can be asked by the active learning approach.

2 http://lact-project.eu

During our experiments, the Java Virtual Machine was allocated 1GB RAM. All experiments were repeated 5 times.

12.5.2 *Results*

The results of our experiments are shown in the Figures below.[3] In all figures, Batch stands for the batch learners while AL stands for the active learners. The numbers in brackets are the sizes of the populations used. The results of the Drugs experiments clearly show that our approach can easily detect simple link specifications. In this experiment, 10 questions were sufficient for the batch and active learning versions of EAGLE to generate link specifications with an F-measure equivalent to the baseline of 99.9% F-measure. The standard deviation lied around 0.1% for all experiments with both batch and active learner.

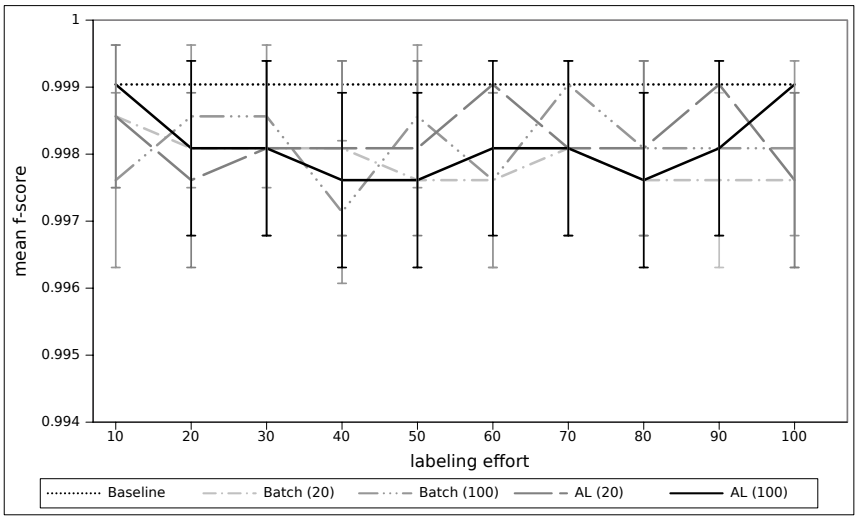


Figure 12.8: Results of the Drugs experiment. Mean F-Measure of five runs of batch and active learner, both using population sizes of 20 and 100 individuals. The baseline is at 99.9% F-measure.

On the more complex Movies experiments, we required 50 inquiries to discover the best link specification that led to an F-measure of 94.1%. This experiment clearly shows the effect of active learning on genetic programming algorithms. While the batch learners fed with any data size tend to diverge significantly across the different experiments as shown by their standard deviation bars, the active learning approaches do not only perform better, they are also more stable as shown by the smaller standard deviation values. Similar results are shown on the most complex and largest dataset at hand, ACM-DBLP.

In addition to being accurate, our approach is very time-efficient. For example, it only required approximately 250ms to run a specification on the first and second datasets when all the data was in memory. On average, it requires less than 700ms on the last dataset. It is important to notice that the features of this dataset include real numbers, which considerably worsen the runtime of link specifications.

Our results suggest that the use of small populations affects the outcome of the learning process significantly, especially when the specification to be learned is

3 Extensive results are available at the LIMES project website at http://limes.sf.net. Accessed June 5, 2012.

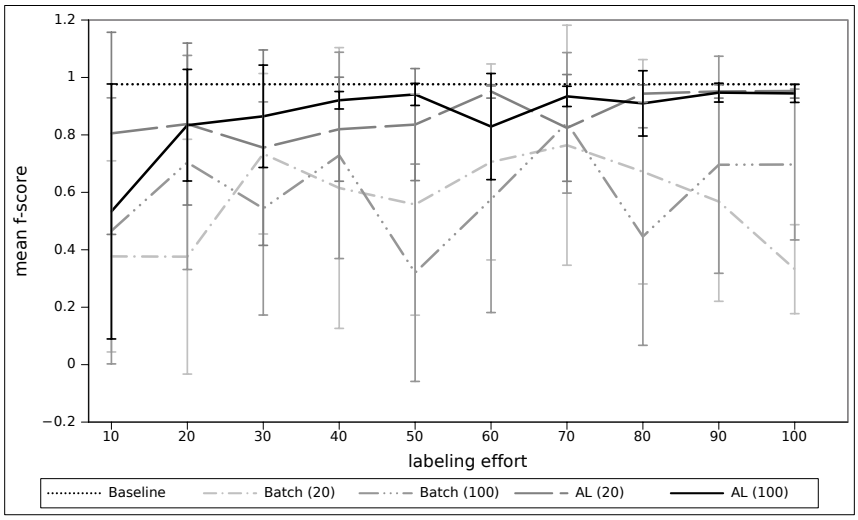


Figure 12.9: Results of the Movies experiment. Mean F-measures of five runs of batch and active learner, both using population sizes of 20 and 100 individuals. The baseline is at 97.6% F-measure.

	EAGLE	FEBRL (SVM)	MARLIN (SVM)	MARLIN (AD-Tree)
F-Measure	97.2%	97.5%	97.6%	96.9%
Runtime	337s	4320s	2196s	1553s

Table 12.2: Comparison of best performances of different machine learning approaches on ACM-DBLP

complex. For example, on the Publications dataset, the learners that rely on solely 20 individuals per generation are up to 9.8% worse than the learners which use populations of 100 individuals. Setting the population to 100 seems to generate sufficiently good results without requiring a large amount of memory. Yet, when trying to link very complex datasets, an even larger setting would be advisable.

12.5.3 *Comparison with other approaches*

As stated above, we chose the ACM-DBLP dataset because it has been used in previous work to compare the accuracy and learning curve of different machine learning approaches for deduplication. As our results show (see Table 12.2), we reach an accuracy comparable to that of the other approaches. One of the main advantages of our approach is that it is considerably more time-efficient that all other approaches. Especially, while we are approximately 3 to 7 times faster than MARLIN, we are more than 14 times faster than FeBRL on this dataset.

So far, only a few other approaches have been developed for learning link specifications from data. RAVEN (Ngonga Ngomo et al., 2011) is an active learning approach that view the learning of link specifications as a classification task. While it bears the advantage of being deterministic, it is limited to learning certain types of classifiers (boolean or linear). Thus, it is only able to learn a subset of the specifications that can be generated by EAGLE. Another genetic programming-based

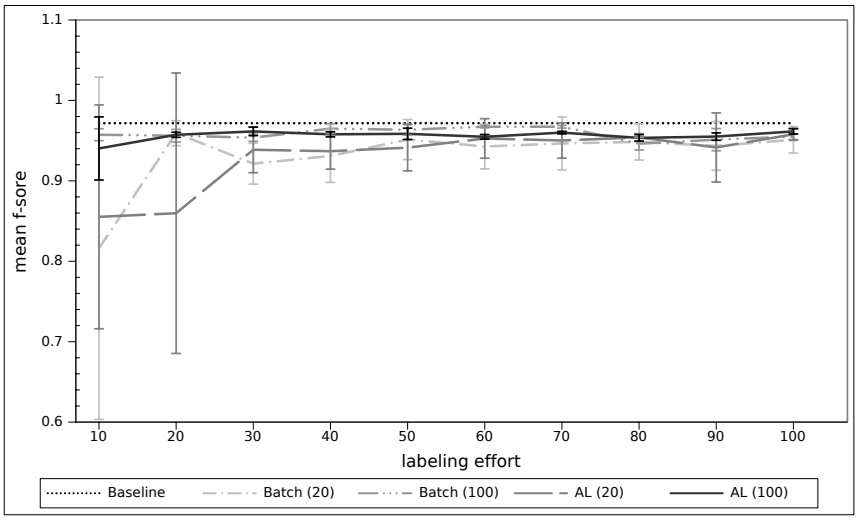


Figure 12.10: Results of the Publications experiment. Mean F-measures of five runs of batch and active learner, both using population sizes of 20 and 100 individuals. The baseline is at 97.2% F-measure.

approach to link discovery is implemented in the SILK framework (Isele and Bizer, 2011). This approach is yet a batch learning approach and it consequently suffers of drawbacks of all batch learning approaches as it requires a very large number of human annotations to learn link specifications of a quality comparable to that of EAGLE.

12.6 CONCLUSION AND FUTURE WORK

In this paper we presented EAGLE, an active learning approach for genetic programming that can learn highly accurate link specifications. We compared EAGLE with its batch learning counterpart. We showed that by using active learning, we can tackle complex datasets with more ease and generate solutions that are more stable (i.e., that display a smaller standard deviation over different runs). We also compared EAGLE with other approaches such as FeBRL and MARLIN on the ACM-DBLP dataset. We showed that for we achieve a similar F-measure while requiring a significantly smaller runtime. We also demonstrated that the runtime of our approach makes it suitable for interactive scenarios. In future work, we will study the effect of different parameterizations in more details. Especially, we will utilize different fitness functions and study the correlation of fitness functions with the overall F-score. Furthermore, we will aim at devising automatic configuration approaches for EAGLE.

13

COALA: CORRELATION-AWARE ACTIVE LEARNING OF LINK SPECIFICATIONS

PREAMBLE

Several active learning approaches were developed and used to facilitate the supervised learning of link specifications. Yet these approaches had not taken the correlation between positive and negative examples into account when requiring labels from their user. In this chapter, we address this drawback by presenting the concept of the correlation-aware active learning of link specifications. The chapter is taken from (Ngonga Ngomo et al., 2013). All algorithms presented in this chapter were designed by the author. He also designed the evaluation, co-wrote the corresponding paper and supervised the remaining works.

13.1 INTRODUCTION

The importance of the availability of links for a large number of tasks such as question answering (Unger et al., 2012) and keyword search (Shekarpour et al., 2011) as well as federated queries has been pointed out often in literature (see, e.g., (Auer et al., 2013a)). Two main problems arise when trying to discover links between datasets or even deduplicate datasets. First, naive solutions to Link Discovery (LD) display a quadratic time complexity (Ngonga Ngomo and Auer, 2011). Consequently, they cannot be used to discover links across large datasets such as DBpedia[1] or Yago.[2] Time-efficient algorithms such as PPJoin+ (Xiao et al., 2008) and $\mathcal{HR}^3$ (Ngonga Ngomo, 2012a) have been developed to address the problem of the a-priori quadratic runtime of LD approaches. While these approaches achieve practicable runtimes even on large datasets, they do not guarantee the quality of the links that are returned by LD frameworks. Addressing this second problem of LD demands the development of techniques that can compute accurate *link specifications* (i.e., aggregations of atomic similarity or distance measures and corresponding thresholds) for deciding whether two resources should be linked. This problem is commonly addressed within the setting of machine learning. While both supervised (e.g., (Ngonga Ngomo and Lyko, 2012)) and unsupervised machine-learning approaches (e.g., (Nikolov et al., 2012)) have been proposed to achieve this goal, we focus on supervised machine learning.

One of the main drawbacks of supervised machine learning for LD lies in the large number of links necessary to achieve both a high precision and a high recall. This intrinsic problem of supervised machine learning has been addressed by relying on active learning (Settles, 2009). The idea here is to rely on *curious classifiers*. These are supervised approaches that begin with a small number of labeled links and then inquire labels for data items that promise to improve their accuracy. Several approaches that combine genetic programming and active learning have been developed over the course of the last couple of years and shown to achieve

1 http://dbpedia.org
2 http://www.mpi-inf.mpg.de/yago-naga/yago/

high F-measures on the deduplication (see e.g., (de Freitas et al., 2010)) and LD (see e.g., (Ngonga Ngomo and Lyko, 2012)) problems. Yet, so far, none of these approaches has made use of the correlation between the unlabeled data items while computing the set of most informative items. In this paper, we address exactly this drawback.

The basic intuition behind this work is that we can provide a better approximation of the real information content of unlabeled data items by taking the similarity of unlabeled items into account. We call this paradigm the correlation-aware active learning of link specifications and dub it COALA. A better approximation should ensure that curious classifiers converge faster. Consequently, we should be able to reduce the number of data items that the user has to label manually. We thus present and evaluate two generic approaches that implement this intuition. Overall, our contributions are as follows:

1. We describe the correlation-aware active learning of link specifications.
2. We present the first two generic approaches that implement this concept. The first is based on graph clustering while the second implements the spreading activation principle.
3. We combine these approaches with the EAGLE algorithm (Ngonga Ngomo and Lyko, 2012) and show in ten different settings that our approaches improve EAGLE's performance with respect to both F-score and standard deviation.

The approaches presented herein were included in the LIMES framework.[3] A demo of the approach can be accessed by using the SAIM interface.[4] The rest of this paper is structured as follows: We first present some of the formal notation necessary to understand this work. In addition, we give some insights into why the inclusion of correlation information can potentially improve the behavior of a curious classifier. Thereafter, we present two approaches that implement the paradigm of including correlation information into the computation of the most informative link candidates. We compare the two approaches with the state of the art in ten different settings and show that we achieve faster convergence and even a better overall performance in some cases. We finally present some related work and conclude.

13.2 PRELIMINARIES

In this section, we present the core of the formal notation used throughout this paper. We begin by giving a brief definition of the problem we address. Then, we present the concept of active learning.

13.2.1 *Link Discovery*

The formal definition of LD adopted herein is similar to that proposed in (Ngonga Ngomo, 2012b). Given a relation R and two sets of instances S and T, the goal of LD is to find the set $M \subseteq S \times T$ of instance pairs (s, t) for which $R(s, t)$ holds. In

3 http://limes.sf.net. Accessed August 1, 2013.
4 http://saim.aksw.org. Accessed August 1, 2013.

most cases, finding an explicit way to compute whether $R(s,t)$ holds for a given pair (s,t) is a difficult endeavor. Consequently, most LD frameworks compute an approximation of M by computing a set $\tilde{M} = \{(s,t) : \sigma(s,t) \geqslant \theta\}$, where σ is a (complex) similarity function and θ is a similarity threshold. The computation of an accurate (i.e., of high precision and recall) similarity function σ can be a very complex task (Isele et al., 2011). To achieve this goal, machine-learning approaches are often employed. The idea here is to regard the computation of σ and θ as the computation of a classifier $C : S \times T \to [-1,+1]$. This classifier assigns pairs (s,t) to the class -1 when $\sigma(s,t) < \theta$. All other pairs are assigned the class $+1$. The similarity function σ and the threshold θ are derived from the decision boundary of C.

13.2.2 *Active Learning of Link Specifications*

Learning approaches based on genetic programming have been most frequently used to learn link specifications (Isele and Bizer, 2011; Ngonga Ngomo and Lyko, 2012; Nikolov et al., 2012). Supervised batch learning approaches for learning such classifiers must rely on large amounts of labeled data to achieve a high accuracy. For example, the genetic programming approach used in (Isele et al., 2012) has been shown to achieve high accuracies when supplied with more than 1000 positive examples. Recent work has addressed this drawback by relying on active learning, which was shown in (Ngonga Ngomo and Lyko, 2012) to reduce the amount of labeled data needed for learning link specifications. The idea behind active learners (also called *curious classifiers* (Settles, 2009)) is to query for the labels of chosen pairs (s,t) iteratively. In the following, we call pairs (s,t) *link candidates*. We denote the count of iterations with t. The function $\mathtt{label} : S \times T \to \{\oplus, \ominus, \otimes\}$ stands for the labeling function and encodes whether a pair (s,t) is (1) known be a positive example for a link (in which case $\mathtt{label}(s,t) = \oplus$), (2) known to be a negative example (in which case $\mathtt{label}(s,t) = \ominus$) or (3) is unclassified (in which case $\mathtt{label}(s,t) = \otimes$). We denote classifiers, similarity functions, thresholds and sets at iteration t by using a superscript notation. For example, the classifier at iteration t is denoted C^t while $\mathtt{label}^t$ stands for the labeling function at iteration t. We call the set $\mathcal{P}^t = \{(s,t) \in S \times T : (\mathtt{label}(s,t) = \otimes) \wedge (C^t(s,t) = +1)\}$ the set *presumed positives*. The set $\mathcal{N}^t$ of *presumed negatives* is defined analogously. If $\mathtt{label}(s,t) = \otimes$, then we call the class assigned by C to (s,t) the *presumed class* of (s,t). When the class of a pair (s,t) is explicit known, we simply use the expression (s,t)*'s class*. The set $C^{+t} = \{(s,t) : C^t(s,t) = +1\}$ is called the set of *positive link candidates* while the set $C^{-t} = \{(s,t) : C^t(s,t) = -1\}$ is called the set of *negative link candidates*. The query for labeled data is carried out by selecting a subset of $\mathcal{P}^t$ with the magnitude k^+ (resp. a subset of $\mathcal{N}^t$ with the magnitude k^-). In the following, we will assume $k = k^+ = k^-$. The selection of the k elements from $\mathcal{P}^t$ and $\mathcal{N}^t$ is carried out by using a function $\mathrm{ifm} : S \times T \to \mathbb{R}$ that can compute how informative a pair (s,t) is for the C^t, i.e., how well the pair would presumably further the accuracy of C^t. We call $\mathcal{I}^{+t} \subseteq \mathcal{P}^t$ (resp. $\mathcal{I}^{-t} \subseteq \mathcal{N}^t$) the set of *most informative positive* (resp. *most informative negative*) link candidates. In this setting, the information content of a pair (s,t) is usually inverse to its distance from the boundary of C^t.

Active learning approaches based on genetic programming adopt a *comittee*-based setting to active learning. Here, the idea is to learn m classifiers $C_1, \ldots, C_m$ concurrently and to have the m classifiers select the sets $\mathcal{I}^-$and $\mathcal{I}^+$. This is usually

carried out by selecting the k unlabeled pairs (s,t) with positive (resp. negative) presumed class which lead to the highest disagreement amongst the classifiers. Several informativeness functions ifm have been used in literature to measure the disagreement. For example, the authors of (Ngonga Ngomo and Lyko, 2012) use the pairs which maximize

$$\text{ifm}(s,t) = (m - \text{pos}(s,t))(m - \text{neg}(s,t)), \tag{13.1}$$

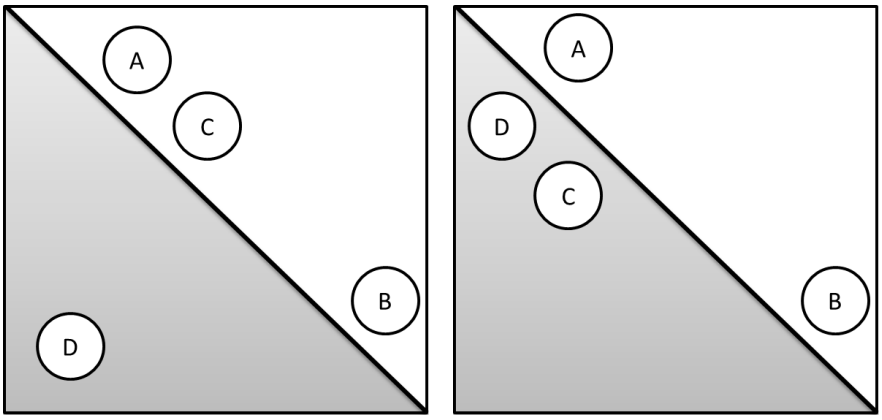
where $\text{pos}(s,t)$ stands for the number of classifiers which assign (s,t) the presumed class $+1$, while $\text{neg}(s,t)$ stands for the number of classifiers which assign (s,t) the class -1. The authors of (Isele et al., 2012) on the other hand rely on pairs (s,t) which maximize the entropy score

$$\text{ifm}(s,t) = H\left(\frac{\text{pos}(s,t)}{m}\right) \text{ where } H(x) = -x\log(x) - (1-x)\log(1-x). \tag{13.2}$$

Note that these functions do not take the correlation between the different link candidates into consideration.

13.3 CORRELATION-AWARE ACTIVE LEARNING OF LINK SPECIFICATIONS

The basic insight behind this paper is that the correlation between the features of the elements of $\mathcal{N}$ and $\mathcal{P}$ should play a role when computing the sets $\mathcal{I}^+$ and $\mathcal{I}^-$. In particular, two main factors affect the information content of a link candidate: its similarity to elements of its presumed class and to elements of the other class. For the sake of simplicity, we will assume that the presumed class of the link candidate of interest is $+1$. Our insights yet hold symmetrically for link candidates whose presumed class is -1.



(a) Intra-correlation (b) Inter-correlation

Figure 13.1: Examples of correlations within classes and between classes. In each subfigure, the gray surface represent $\mathcal{N}$ while the white surface stands for $\mathcal{P}$. The oblique line is $\mathcal{C}$'s boundary.

Let $A = (s_A, t_A), B = (s_B, t_B) \in \mathcal{P}$ to be two link candidates which are equidistant from $\mathcal{C}$'s boundary. Consider Figure 13.1a, where $\mathcal{P}$= {A, B, C} and $\mathcal{N}$= {D}. The link candidate B is on on average most distant from any other elements of $\mathcal{P}$. Thus, it is more likely to be a statistical outlier than A. Hence, making a classification

error on B should not have the same impact as an erroneous classification of link candidate A, which is close to another presumably positive link candidate, C. Consequently, B should be considered less informative than A. Approaches that make use of this information are said to exploit the *intra-class correlation*. Now, consider Figure 13.1b, where $\mathcal{P}$= {A, B} and $\mathcal{N}$= {C, D}. While the probability of A being an outlier is the same as B's, A is still to be considered more informative than B as it is located closer to elements of $\mathcal{N}$ and can thus provide more information on where to set the classifier boundary. This information is dubbed *inter-class correlation*. Several approaches that make use of these two types of correlations can be envisaged. In the following, we present two approaches for these purposes, of which one is based on graph clustering and the other on spreading activation combined with weight decay.

13.4 APPROACHES

In this section, we present two approaches for the correlation-aware active learning of link specifications. The first makes use of intra-class correlations and relies on graph clustering. The second approach relies on the spreading activation principle in combination with weight decay. We assume that the complex similarity function σ underlying $\mathcal{C}$ is computed by combining n atomic similarity functions $\sigma_1, \ldots, \sigma_n$. This combination is most commonly carried out by using metric operators such as min, max or linear combinations.[5] Consequently, each link candidate (s, t) can be described by a vector $(\sigma_1(s,t), \ldots, \sigma_n(s,t)) \in [0,1]^n$. We define the *similarity of link candidates* $\mathrm{sim} : (S \times T)^2 \to [0,1]$ to be the inverse of the Euclidean distance in the space spawned by the similarities σ_1 to σ_n. Hence, the similarity of two link candidates (s, t) and (s', t') is given by:

$$\mathrm{sim}((s,t),(s',t')) = \frac{1}{1 + \sqrt{\sum_{i=1}^{n} (\sigma_i(s,t) - \sigma_i(s',t'))^2}}. \tag{13.3}$$

Note that we added 1 to the denominator to prevent divisions by 0.

13.4.1 *Graph Clustering*

The basic intuition behind using clustering for COALA is that groups of very similar link candidates can be represented by a single link candidate. Consequently, once a representative of a group has been chosen, all other elements of the group become less informative. An example that illustrates this intuition is given in Figure 13.2. We implemented COALA based on clustering as shown in Algorithm 13.1. In each iteration, we begin by first selecting two sets $\mathcal{S}^+ \subseteq \mathcal{P}$ resp. $\mathcal{S}^- \subseteq \mathcal{N}$ that contain the positive resp. negative link candidates that are most informative for the classifier at hand. Formally, $\mathcal{S}^+$ fulfils

$$\forall x \in \mathcal{S}^+ \; \forall y \in \mathcal{P}, y \notin \mathcal{S}^+ \to \mathrm{ifm}(y) \leqslant \mathrm{ifm}(x). \tag{13.4}$$

The analogous equation holds for $\mathcal{S}^-$. In the following, we will explain the further steps of the algorithm for $\mathcal{S}^+$. The same steps are carried out for $\mathcal{S}^-$.

5 See (Ngonga Ngomo, 2012b) for a more complete description of a grammar for link specifications.

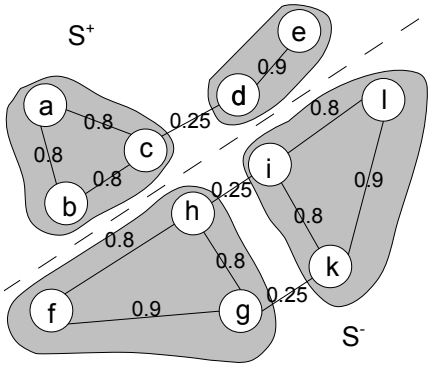


Figure 13.2: Example of clustering. One of the most informative single link candidate is selected from each cluster. For example, d is selected from the cluster $\{d, e\}$.

First, we compute the similarity of all elements of $\mathcal{S}^+$ by using the similarity function shown in Equation 13.3. In the resulting similarity matrix, we set all elements of the diagonal to 0. Then, for each $x \in \mathcal{S}^+$, we only retain a fixed number ec of highest similarity values and set all others to 0. The resulting similarity matrix is regarded as the adjacency matrix of an undirected weighted graph $G = (V, E, sim)$. G's set of nodes V is equal to $\mathcal{S}^+$. The set of edges E is a set of 2-sets[6] of link candidates. Finally, the weighted function is the similarity function sim. Note that ec is the minimal degree of nodes in G.

In a second step, we use the graph G as input for a graph clustering approach. The resulting clustering is assumed to be a partition $\mathcal{V}$ of the set V of vertices of G. The informativeness of partition $V_i \in \mathcal{V}$ is set to $\max_{x \in V_i} \text{ifm}(x)$. The final step of our approach consists of selecting the most informative node from each of the k most informative partitions. These are merged to generate $\mathcal{I}^+$, which is sent as query to the oracle. The computation of $\mathcal{I}^-$ is carried out analogously. Note that this approach is generic in the sense that it can be combined with any graph clustering algorithm that can process weighted graphs as well as with any informativeness function ifm. Here, we use BorderFlow (Ngonga Ngomo and Schumacher, 2009) as clustering algorithm because (1) it has been used successfully in several other applications (Morsey et al., 2011, 2012; Ngonga Ngomo, 2010) and (2) it is parameter-free and does not require any tuning.

13.4.2 *Spreading Activation with Weight Decay*

The idea behind spreading activation with weight decay (WD) is to combine the intra- and inter-class correlation to determine the informativeness of each link candidate. Here, we begin by computing the set $\mathcal{S} = \mathcal{S}^+ \cup \mathcal{S}^-$, where $\mathcal{S}^+$ and $\mathcal{S}^-$ are described as above. Let s_i and s_j be the i^{th} and j^{th} elements of $\mathcal{S}$. We then compute the quadratic similarity matrix $\mathcal{M}$ with entries $m_{ij} = sim(s_i, s_j)$ for $i \neq j$ and 0 else. Note that both negative and positive link candidates belong to $\mathcal{S}$. Thus, $\mathcal{M}$ encodes both inter- and intra-class correlation. In addition to $\mathcal{M}$, we compute the activation vector $\mathcal{A}$ by setting its entries to $a_i = \text{ifm}(s_i)$. In the following, $\mathcal{A}$ is considered to be a column vector. The spreading of the activation with weight decay is then carried out as shown in Algorithm 13.2.

6 A n-set is a set of magnitude n.

Algorithm 13.1 COALA based on Clustering

Require: `mappingSet` (set of mappings)
Require: `exampleCount` (number of examples)
Require: `edgesPerNode` (maximal number of edges per node)
$\mathcal{S}^-$:=get closest negative mappings($mappingSet$)
$\mathcal{S}^+$:= get closest positive mappings($mappingSet$)
`clusterSet` := $\emptyset$
for $set \in \{\mathcal{S}^-, \mathcal{S}^+\}$ **do**
  G := buildGraph(set,edgesPerNode)
  $clusterSet \leftarrow clustering(G)$
  $visitedClusters := \emptyset$, $addedElements$:=0
  $sortedMappingList$:= sortingByDistanceToClassfier($mappingSet$)
  while $addedElements \neq exampleCount$ **do**
    (s, t):= next($sortedMappingList$)
    $partition$:=getPartition((s, t))
    if $partition \notin visitedClusters$ **then**
      oracleList:=add((s, t))
      $addedElements++$
      $visitedClusters$:=addCluster($partition$)
    end if
  end while
end for
return `oracleList`

In a first step, we normalize the activation vector $\mathcal{A}$ to ensure that the values contained therein do not grow indefinitely. Then, in a second step, we set $\mathcal{A} = \mathcal{A} + \mathcal{M} \times \mathcal{A}$. This has the effect of propagating the activation of each s to all its neighbours according to the weights of the edges between s and its neighbours. Note that elements of $\mathcal{S}^+$ that are close to elements of $\mathcal{S}^-$ get a higher activation than elements of $\mathcal{S}^+$ that are further away from $\mathcal{S}^-$ and vice-versa. Moreover, elements at the centre of node clusters (i.e., elements that are probably no statistical outliers) also get a higher activation than elements that are probably outliers. The idea behind the weight decay step is to update the matrix by setting each m_{ij} to m_{ij}^r, where $r > 1$ is a fix exponent. This is the third step of the algorithm. Given that $\forall i \forall j\ m_{ij} \leqslant 1$, the entries in the matrix get smaller with time. By these means, the amount of activation transferred across long paths is reduced. We run this three-step procedure iteratively until all non-1 entries of the matrix are less or equal to a threshold $\epsilon = 10^{-2}$. The k elements of $\mathcal{S}^+$ resp. $\mathcal{S}^-$ with maximal activation are returned as $\mathcal{I}^+$resp. $\mathcal{I}^-$. In the example shown in Figure 13.3, while all nodes from $\mathcal{S}^+$ and $\mathcal{S}^-$ start with the same activation, two nodes get the highest activation after only 3 iterations.

13.5 EVALUATION

The goal of our evaluation was to study the improvement in F-score achieved by integrating the approaches presented above with a correlation-unaware approach to learning link specifications. We chose to use EAGLE (Ngonga Ngomo and Lyko, 2012), an approach based on genetic programming which was shown to outper-

Algorithm 13.2 COALA based on Weight Decay

Require: $mappingSet$ (set of mappings)
Require: r (exponent of the fixpoint)
Require: $exampleCount$ (number of examples)
Ensure: $oracleList$ (list of mappings)
1: $\mathcal{M}$:= buildAdjacenceMatrix($mappingSet$)
2: $\mathcal{A}$:= buildActivationVector($mappingSet$)
3: **while** $\exists m_{ij} : m_i j \neq 1 \wedge m_{ij} > \epsilon$ **do**
4: **for all** $m_{ij} \in \mathcal{M} | m_{ij} \neq 1 : m_{ij} \leqslant \epsilon$ **do**
5: $\mathcal{A} := \mathcal{A} / max_{\mathcal{A}}$
6: $\mathcal{A} := \mathcal{A} + \mathcal{M} \times \mathcal{A}$
7: $\mathcal{M} := (\forall m_{ij} \in \mathcal{M} : m_{ij} := m_{ij}^{r})$
8: **end for**
9: **end while**
10: **return** $oracleList$:= getMostActivatedMapping($\mathcal{A}$,$exampleCount$)

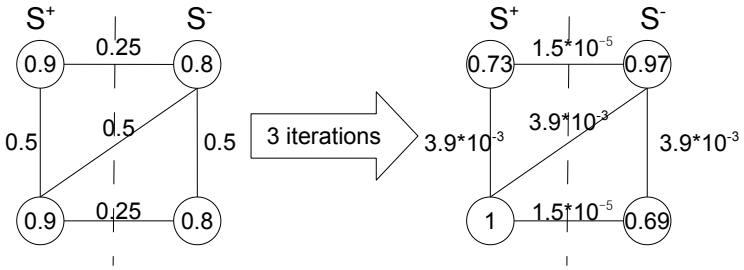


Figure 13.3: Example of weight decay. Here r was set to 2. The left picture shows the initial activations and similarity scores while the right picture shows the results after 3 iterations. Note that for the sake of completeness the weights of the edges were not set to 0 when they reached ϵ.

form batch learning. We ran a preliminary experiment on one dataset to determine good parameter settings for the combination of EAGLE and clustering (CL) as well as the combination EAGLE and weight decay (WD). Thereafter, we compared the F-score achieved by EAGLE with that of CL and WD in ten different settings.

13.5.1 *Experimental Setup*

Throughout our experiments, we set both mutation and crossover rates to 0.6. Individuals were given a 70% chance to get selected for reproduction. The population sizes were set to 20 and 100. We set $k = 5$ and ran our experiments for 10 iterations. Between each iteration we evolved the populations for 50 generations. We ran our experiments on two real-world datasets and three synthetic datasets. The synthetic datasets consisted of the datasets from the OAEI 2010 benchmark.[7] The real-world datasets consisted of the ACM-DBLP and Abt-Buy datasets, which were extracted from websites or databases (Köpcke et al., 2009).[8] The ACM-DBLP dataset consists

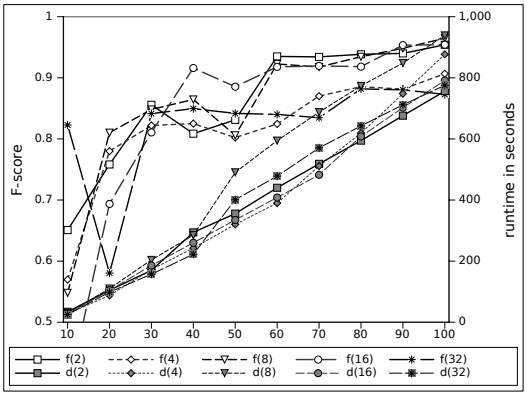
7 http://oaei.ontologymatching.org/2010/
8 http://dbs.uni-leipzig.de/en/research/projects/object_matching/fever/benchmark_datasets_for_entity_resolution

of 2,617 source and 2,295 target publications with 2,224 links between them. The Abt-Buy dataset holds 1,092 links between 1,081 resp. 1,092 products. Note that this particular dataset is both noisy and incomplete. All non-RDF datasets were transformed into RDF and all string properties were set to lower case by default as the only preprocessing step. Given that genetic programming is non-deterministic, all results presented below are the means of 5 runs. Each experiment was ran on a single thread of a server running JDK1.7 on Ubuntu 10.0.4 and was allocated maximally 2GB of RAM. The processors were 2.0GHz Quadcore AMD Opterons.

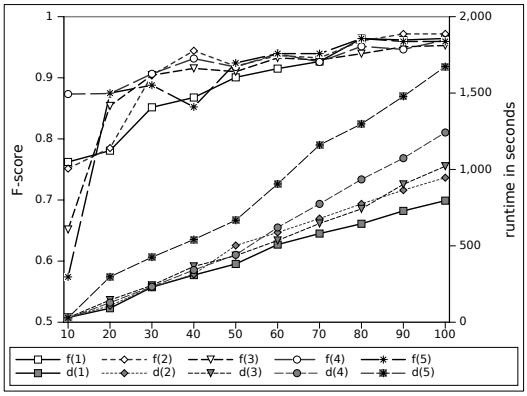
13.5.2 *Results*

13.5.2.1 *Parametrization of WD and CL*

In a preliminary series of experiments we tested for a good parametrization of both WD and CL. For this purpose we ran both approaches on the DBLP-ACM dataset using 5 different values for the r exponent for weight decay and the clustering ec parameter. The tests were ran with a population of 20, $r = \{2, 4, 8, 16, 32\}$ and $ec = \{1, 2, 3, 4, 5\}$. Figure 13.4a and Figure 13.4b show the results of achieved F-scores and runtimes. In both plots $f(p)$ and $d(p)$ denote the F-score and runtime of the particular method using the p parameter. Figure 13.4a suggests that $r = 2$



(a) Testing different r parameter of the weight decay computation



(b) Testing different edge count constraints ec for the clustering method

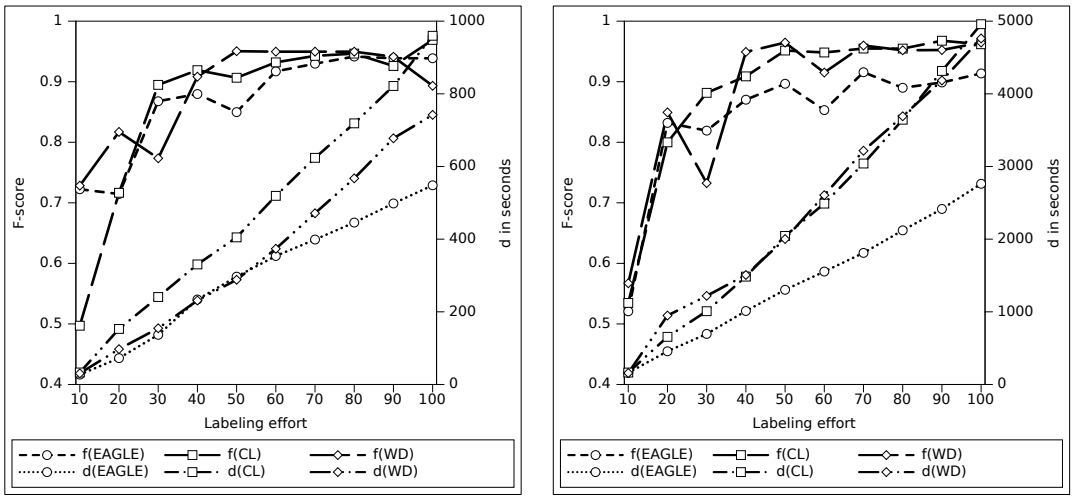
Figure 13.4: Testing different r and ec parameter for both approaches on the DBLP-ACM dataset. $f(p)$ denotes the F-score achieved with the method using the parameter p, while $d(p)$ denotes the required run time.

leads to a good accuracy (especially for later inquiries) while requiring moderate computation resources. Similarly, $r = 16$ promises fast convergence and led to better results in the fourth and fifth iterations. Still, we chose $r = 2$ for all experiments due to an overall better performance. The test for different ec parameters led us to use an edge limit of $ec = 3$. This value leads to good results with respect to both accuracy and runtime as Figure 13.4b suggests.

13.5.2.2 *Runtime and F-score*

Figure 13.5 - Figure 13.9 show the results of both our approaches in comparison to the EAGLE algorithm. A summary of the results achieved by the approaches

is given in Table 13.1. Most importantly, our results suggest that using correlation information can indeed improve the F-score achieved by curious classifiers. The average of the results achieved by the approaches throughout the learning process (left group of results in Table 13.1) shows that already in average our approaches outperform EAGLE in 9 from 10 settings. A look at the final F-scores achieved by the approaches show that one of the approaches WD and CL always outperform EAGLE both with respect to the average F-score and the standard deviation achieved across the 5 runs except on the Restaurant dataset for a population size of 100, where the results of CL and EAGLE are the same. This leads us to conclude that the intuition underlying this paper is indeed valid. Interestingly, the experiments presented herein do not allow declaring CL superior to WD or vice-versa. While CL performs better on the small population, WD catches up on larger populations and outperform CL in 3 of 5 settings. An explanation for this behaviour could lie in WD taking more information into consideration and thus being more sensible to outliers than CL. A larger population size which reduces the number of outliers would then be better suited to WD. This explanation is yet still to be proven in larger series of experiments and in combination with other link discovery approaches such as RAVEN. Running WD and CL is clearly more time-demanding than simply running EAGLE. Still the overhead remains within acceptable boundaries. For example, while EAGLE needs approx. 2.9s for 100 individuals on the Abt-Buy dataset while both WD and CL require 3.4s (i.e., 16.3% more time).



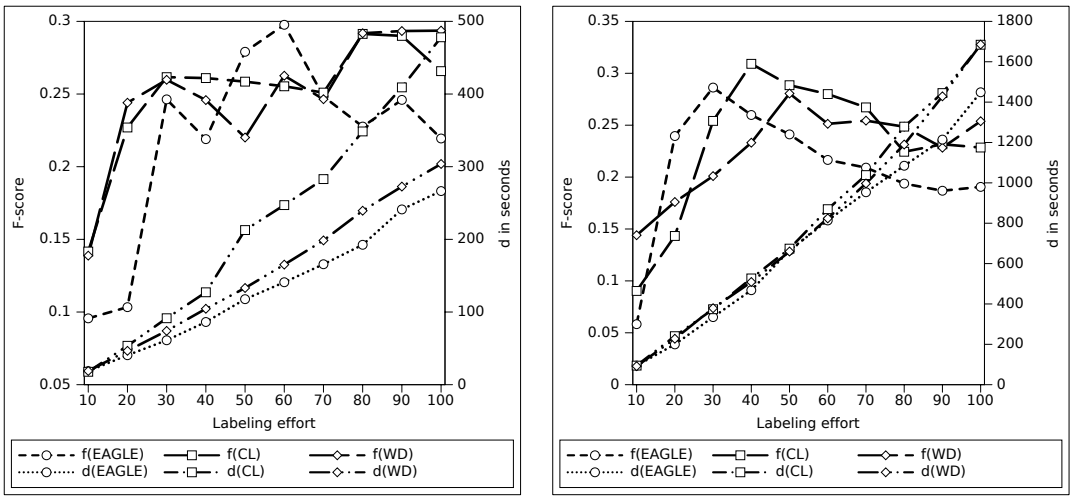
(a) Population = 20 individuals

(b) Population = 100 individuals

Figure 13.5: F-score and runtime on the ACM-DBLP dataset. f(X) stands for the F-score achieved by algorithm X, while d(X) stands for the total duration required by the algorithm.

13.6 RELATED WORK

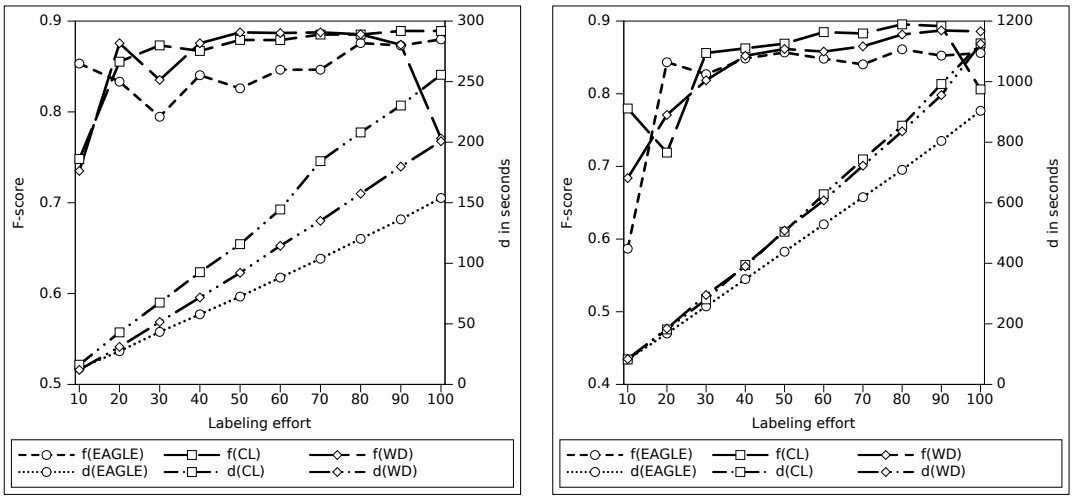
The number of LD approaches has proliferated over the last years. Herein, we present a brief overview of existing approaches (see (Isele et al., 2012; Ngonga Ngomo, 2012a) for more extensive presentations of the state of the art). Overall, two main problems have been at the core of the research on LD. First, the time complexity of LD was addressed. In (Ngonga Ngomo and Auer, 2011), an approach



(a) Results of the Abt-Buy dataset using a population of 20 individuals

(b) Results of the Abt-Buy dataset using a population of 100 individuals

Figure 13.6: F-score and runtime on the Abt-Buy dataset



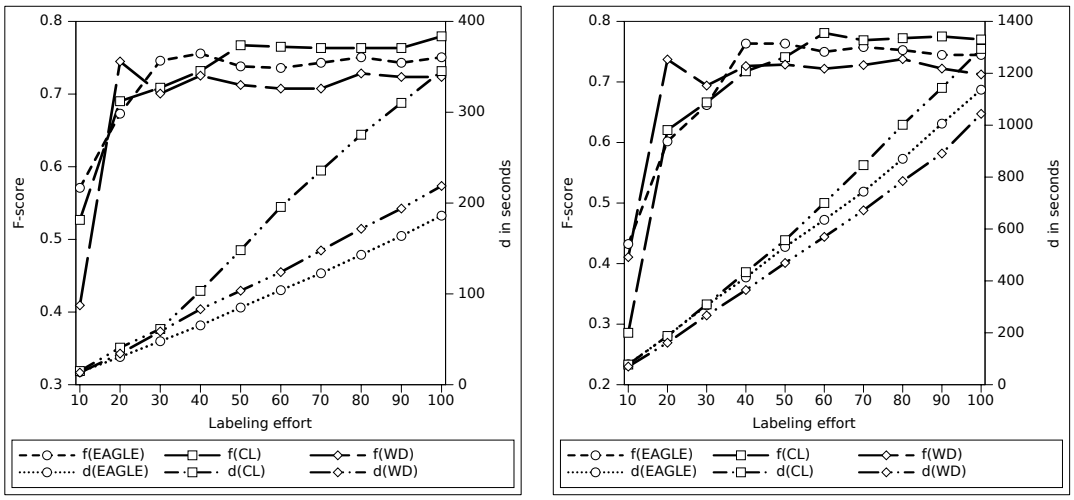
(a) Results of the Person1 dataset using a population of 20 individuals.

(b) Results of the Person1 dataset using a population of 100 individuals.

Figure 13.7: F-score and runtime on the OAEI 2010 Person1 dataset

based on the Cauchy-Schwarz inequality was used to reduce the runtime of LD processes based on metrics. The approach $\mathcal{HR}^3$ (Ngonga Ngomo, 2012a) rely on space tiling in spaces with measures that can be split into independent measures across the dimensions of the problem at hand. Especially, $\mathcal{HR}^3$ was shown to be the first approach that can achieve a relative reduction ratio r' less or equal to any given relative reduction ratio $r > 1$. Concepts from the deduplication research field were also employed for LD. For example, standard blocking approaches were implemented in the first versions of SILK[9] and later replaced with MultiBlock (Isele et al., 2011), a lossless multi-dimensional blocking technique. KnoFuss (Nikolov

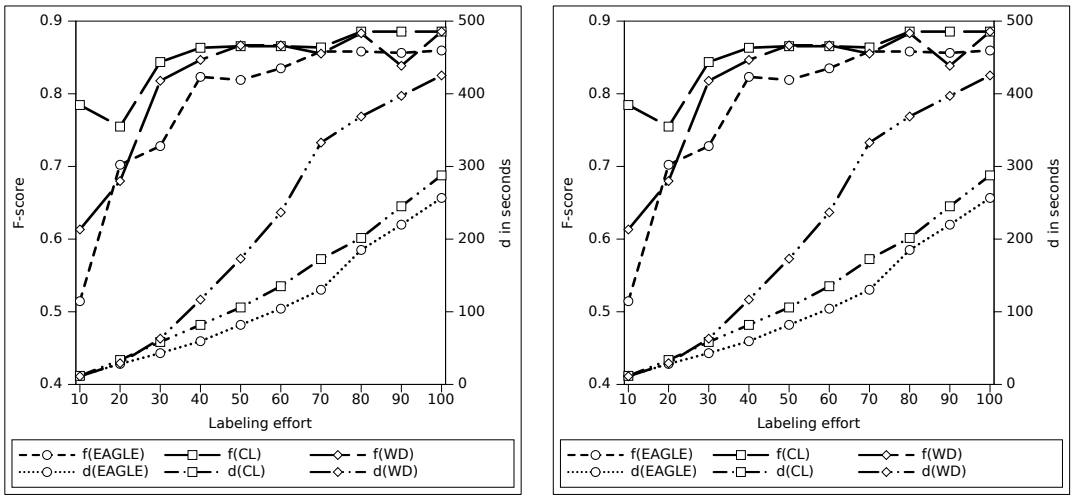
9 http://wifo5-03.informatik.uni-mannheim.de/bizer/silk/



(a) Results of the Person2 dataset using a population of 20 individuals.

(b) Results of the Person2 dataset using a population of 100 individuals.

Figure 13.8: F-score and runtime on the OAEI 2010 Person2 dataset



(a) Results of the Restaurant dataset using a population of 20 individuals.

(b) Results of the Restaurant dataset using a population of 100 individuals.

Figure 13.9: F-score and runtime on the OAEI 2010 Restaurant dataset

et al., 2012) also implements blocking techniques to achieve acceptable runtimes. Moreover, time-efficient string comparison algorithms such as PPJoin+ (Xiao et al., 2008) were integrated into the hybrid framework LIMES (Ngonga Ngomo, 2012b). Other LD frameworks can be found in the results of the ontology alignment evaluation initiative (Euzenat et al., 2011).

The second problem that was addressed is the complexity of link specifications. Although unsupervised techniques were newly developed (see, e.g., (Nikolov et al., 2012)), most of the approaches developed so far abide by the paradigm of supervised machine learning. For example, the approach presented in (Isele and Bizer, 2011) relies on large amounts of training data to detect accurate link specification using genetic programming. RAVEN (Ngonga Ngomo et al., 2011) is (to the best

	Average values			Final values		
DataSet	**EAGLE**	**WD**	**CL**	**EAGLE**	**WD**	**CL**
Abt	0.22± 0.06	**0.25 ± 0.07**	0.25 ± 0.08	0.22± 0.05	**0.29 ± 0.03**	0.27 ± 0.05
DBLP	0.87± 0.1	**0.89± 0.09**	0.87± 0.08	0.94± 0.02	0.89± 0.13	**0.97± 0.0**
Person1	0.85± 0.05	0.85± 0.06	**0.87± 0.03**	0.88± 0.02	0.77± 0.25	**0.89± 0.01**
Person2	0.72± 0.05	0.69± 0.11	**0.73± 0.08**	0.75± 0.02	0.72± 0.09	**0.78± 0.0**
Rest.	0.79± 0.13	0.82± 0.08	**0.85± 0.05**	0.51± 0.36	0.61± 0.28	**0.78± 0.01**
Abt	0.21 ± 0.06	0.23± 0.07	**0.23± 0.05**	0.19 ± 0.04	**0.25± 0.04**	0.23± 0.04
DBLP	0.87± 0.1	0.89± 0.09	**0.89± 0.08**	0.91± 0.03	**0.96± 0.01**	0.96± 0.02
Person1	0.82± 0.05	**0.84± 0.07**	**0.84± 0.07**	0.86± 0.02	**0.89± 0.01**	0.81± 0.18
Person2	**0.7± 0.09**	0.69± 0.1	0.69± 0.07	0.74± 0.03	0.71± 0.08	**0.77± 0.03**
Rest.	0.81± 0.11	0.82± 0.06	**0.85± 0.03**	**0.89± 0.0**	0.86± 0.02	**0.89± 0.0**

Table 13.1: Comparison of average F-scores achieved by EAGLE, WD and CL. The top section of the table shows the results for a population size of 20 while the bottom part shows the results for 100 individuals. Best scores are in bold font. Abt stands for Abt-Buy, DBLP for DBLP-ACM and Rest. for Restaurants.

of our knowledge) the first active learning technique for LD. The approach was implemented for linear or Boolean classifiers and shown to require a small number of queries to achieve high accuracy. While the first active genetic programming approach was presented in (de Freitas et al., 2010), similar approaches for LD were developed later (Isele et al., 2012; Ngonga Ngomo and Lyko, 2012). Still, none of the active learning approaches for LD presented in previous work made use of the similarity of unlabelled link candidates to improve the convergence of curious classifiers. Yet, works in other research areas have started considering the combination of active learning with graph algorithms (see e.g., (Bodó et al., 2011)).

13.7 CONCLUSION

We presented the first generic LD approaches that make use of the correlation between positive and negative link candidates to achieve a better convergence. The first approach is based on clustering and only makes use of correlations within classes while the second algorithm makes use of both correlations within and between classes. We compared these approaches on 5 datasets and showed that we achieve better F-scores and standard deviations than the EAGLE algorithm. Thus, in future work, we will integrate our approach into other algorithms such as RAVEN. Moreover, we will measure the impact of the graph clustering algorithm utilized in the first approach on the convergence of the classifier. Our experimental results showed that each of the approaches we proposed has its pros and cons. We will thus explore combinations of WD and CL.

14 UNSUPERVISED LEARNING OF LINK SPECIFICATIONS

As shown in previous chapters, genetic programming can be used for link discovery. The questions underlying this chapter are drawn from the evaluation paper presented in (Ngonga Ngomo and Lyko, 2013). The study was designed and partly implemented by the author, who also co-wrote the paper.

14.1 INTRODUCTION

Over the last years, the importance of Link Discovery (LD) as a research topic has increased significantly. This increase was upheld mainly by the ever-growing size of the Linked Data Web and the scalability and accuracy requirements it brings about. The creation of links between knowledge bases, one of the most important steps in the realization of the vision of the Linked Data Web has profited from this boost of research and seen the development of several LD frameworks and approaches (Ngonga Ngomo and Auer, 2011; Hogan et al., 2010; Sleeman and Finin, 2010; Papadakis et al., 2011; Isele et al., 2011). Two main research focuses played a role so far: (1) the determination of time-efficient algorithms (Isele et al., 2011; Ngonga Ngomo and Auer, 2011; Ngonga Ngomo, 2011) for LD and (2) the development of approaches for the efficient computation of link specifications (also called linkage rules) (Ngonga Ngomo et al., 2011; Isele and Bizer, 2011; Nikolov et al., 2012; Ngonga Ngomo and Lyko, 2012). In most cases, supervised machine learning approaches were used to tackle the second challenge of LD. Approaches developed so far include batch learning using genetic programming (Isele et al., 2011), the combination of active learning and of linear and Boolean classifiers (Ngonga Ngomo et al., 2011) as well as the combination of active learning and genetic programming (Ngonga Ngomo and Lyko, 2012). In addition, unsupervised approaches for learning link specifications have been recently developed (Nikolov et al., 2012). While all these approaches have been shown to achieve good results, unsupervised approaches obviously trump batch and active learning approaches as they do not require any feedback from the user and can still achieve remarkably good performance. In addition, genetic programming approaches yield the central advantage of being able to exploit the whole spectrum of the link specification grammar provided by the framework in which they were implemented. So far, unsupervised approaches to the discovery of link specifications have only been tested with artificially generated benchmark data and low-noise datasets. Moreover, no deterministic approach for the unsupervised discovery of link specifications has been presented so far, although deterministic approaches such as those presented in (Ngonga Ngomo et al., 2011) counterbalance their limitations in expressiveness by being clearly more time-efficient than approaches based on genetic programming.

The aim of this chapter is to experimentally examine the unsupervised discovery of link specifications with respect to two main questions:

1. *Are deterministic approaches able to achieve results comparable to those of genetic approaches?* To address this question, we extended the approach presented

in (Ngonga Ngomo and Lyko, 2012) and devised an approach for the unsupervised learning of Boolean and linear classifiers which is loosely based on the RAVEN approach (Ngonga Ngomo et al., 2011). We refrained from reusing the approach presented in (Nikolov et al., 2012) as one of the pseudo-measures we rely on was designed especially to work well with this approach. Consequently, using it could have led to a bias in our results.

2. *How well are pseudo-F-measures suited for unsupervised discovery performed on synthetic and real datasets?* Here, we compared the results achieved by the approaches above on the three OAEI 2010 datasets[1] and on three datasets extracted from real data.[2] In addition to the pseudo-F-measure described in (Nikolov et al., 2012) (which we dub $\mathcal{F}_u^\beta$), we devised a supplementary pseudo-F-measure $\mathcal{F}_d^\beta$ which relies more on the standard definition of the F_β-measure. We performed a correlation analysis of the values of $\mathcal{F}_u^\beta$, $\mathcal{F}_d^\beta$ and the F_1 measure and detected a surprisingly different behaviour of these measures across our two groups of datasets.

The rest of this paper is structured as follows: We first give an overview of the approaches and measures we used for our experiments. We then present the results of our experimental setup as well as the results of our experiments. For the sake of reproducibility, we chose to use freely available datasets and made the approaches presented herein freely available at the project website.[3] We conclude with a summary of the implications of our results for the LD community.

14.2 APPROACHES

In general, a link specification is a classifier C that assigns each element of the set $S \times T$ to one of the classes of $Y = \{+1, -1\}$, where S is called the set of source instances, while T is the set of target instances. $(s,t) \in S \times T$ is considered by C to be a correct link when $C(s,t) = +1$. Otherwise, (s,t) is considered not be a potential link. We will assume that the classifier C relies on a complex similarity function σ which consists of a combination of atomic similarity measure σ_i. Each of the atomic similarity measures is associated with a parameter ω_i, which is used in main cases as threshold or weight for σ_i. Supervised approaches to the computation of link specifications use labelled training data $L \subseteq S \times T \times Y$ to maximize an objective function such as the distance from the labelled data items to the boundary of the classifier in the case of Support Vector Machines (Cristianini and Ricci, 2008). The idea behind unsupervised approaches to learning link specifications is not to utilize any training data (i.e., $L = \emptyset$). Instead, they aim to optimize an objective function $\mathcal{F}$. In the following, we present the non-deterministic and the deterministic approaches we utilized in our experiments. We then present two different objective functions that are based on the well-know F_β-measure. These functions build the basis for our evaluation.

1 Freely available at http://oaei.ontologymatching.org/2010/.
2 Freely available at http://dbs.uni-leipzig.de/en/research/projects/object_matching/fever/benchmark_datasets_for_entity_resolution.
3 http://saim.sf.net

14.2.1 *Non-Deterministic Approach*

The non-deterministic approach we evaluated is based on the EAGLE approach presented in (Ngonga Ngomo and Lyko, 2012) and implemented in the LIMES framework (Ngonga Ngomo et al., 2011). The approach was modified as described in Algorithm 14.1. We begin by generating a random population of n individuals. Let G^t be the population at the iteration t. To evolve a population to the generation G^{t+1}, the fitness of each individuals $g^t \in G^t$ is computed. For this purpose, the mapping $M(g^t)$ generated by g^t is evaluated and the value $\mathcal{F}(M(g^t))$ is assigned to g^t. These fitness values build the basis for selecting individuals for the genetic operator *reproduction*. EAGLE uses a tournament setting between two selected individuals to decide which one is copied to the next generation g^{t+1}. On randomly selected individuals the operator *mutation* is applied according to a probability called the *mutation rate*. A mutation can affect an individual in three different ways: First, it can alter the thresholds used by the individual. Second, a mutation can alter the property values that are compared by one of the atomic measures on which the classifier relies. Finally, mutations can modify the measures included in the individuals. The third genetic operator, *crossover*, operates on two parent individuals and builds a new offspring by swapping two random sub-trees of the parent genotypes. The application of these operators is carried out iteratively until a maximal number of iterations is reached. The result of this process is the mapping M that is returned by the best individual. M is the postprocessed as follows: Each $s \in S$ such that $\exists t \in T : (s,t) \in M$ is mapped to $\arg\max_{t \in T} \sigma(s,t)$. The postprocessed mapping is finally returned.

Algorithm 14.1 Overview of the EAGLE Algorithm

Require: Sets of instances S and T, size of population, number of iterations
  Get property mapping (S, T)
  Generate initial population
  repeat
    Compute $\mathcal{F}$ for all individuals.
    Apply genetic operators to population
  until Number of iterations is reached
  return Overall fittest individual

14.2.2 *Deterministic Approaches*

Linear and Boolean classifier have been shown in previous work (Ngonga Ngomo et al., 2011) to also achieve good results on the task of LD. Both types of classifiers can be characterized by a similarity function σ which depends on similarity measures σ_i and parameters ω_i. Linear classifiers $\mathcal{L}$ classify (s,t) as belonging to $+1$ iff $\sum_{i=1}^{n} \omega_i \sigma_i(s,t) \geqslant 1$. For Boolean classifiers $\mathcal{B}$, the inequality $\bigwedge_{i=1}^{n} \sigma_i(s,t) \geqslant \omega_i$ must be fulfilled by the pair (s,t) for it belong to $+1$. In both cases, a classifier can be encoded by the vector $\Omega = (\omega_1, \ldots, \omega_n)$. Determining the best $\mathcal{L}$ or $\mathcal{B}$ would require testing all possible combinations of values $\omega_i \in [0,1]$, which would be impracticable. The idea behind our algorithm EUCLID (Efficient and Unsupervised Classification for Link Discovery) is to reduce the number of configurations

Algorithm 14.2 Overview of the EUCLID Algorithm

Require: Specification of the datasets S and T
Require: $\Delta_\omega \in]0, 1[$
Require: $\gamma > 1$, $\theta > 0$ with $(\gamma, \theta) \in \mathbb{N}^2$
  Get property mapping (S, T)
  bestClassifier = $(1, \ldots, 1)$
  $\Omega = \emptyset$
  for $k = 0 \to \lfloor 1/\Delta_\omega \rfloor$ **do**
    $\Omega = \Omega \cup \{1 - k\Delta_\omega\}$
  end for
  for $i = 1 \to n$ **do**
    $\sigma_i^{max} = \underset{\omega \in \Omega, \sigma_i \in \Sigma}{\arg\max}\ \mathcal{F}(\sigma_i \geqslant \omega)$, $\omega_i^{min} = 0$, $\omega_i^{max} = 1$
  end for
  for $iterations = 1 \to \theta$ **do**
    $\Delta = \frac{|\omega_i^{max} - \omega_i^{min}|}{\gamma}$
    $C = \underset{i=1}{\overset{n}{\arg\max}}\, \mathcal{F}(\omega_i^{min} + k_i\Delta)$
    if $\mathcal{F}(C) < \mathcal{F}(\text{bestClassifier})$ **then**
      C = bestClassifier
    else
      bestClassifier = C
    end if
    for $j = 1 \to n$ **do**
      $\omega_j^{min} = \max(0, \zeta_j)$, $\omega_j^{max} = \min(1, \zeta_j)$
    end for
  end for
  return bestClassifier

that must be tested by applying a search of the search space whose granularity increases gradually as described in Algorithm 14.2: We first begin by detecting the right similarity measure for each pair of properties. To achieve this goal, we use the similarity $\sigma_i^{max} = \underset{\omega \in \Omega, \sigma_i \in \Sigma}{\arg\max}\ \mathcal{F}(\sigma_i \geqslant \omega)$ for each pair of properties, where $\mathcal{F}(\sigma_i \geqslant \omega)$ is the value of the pseudo-F-measure (PFM) of the classifier C_0 such that $C_0(s,t) = +1 \iff \sigma_i(s,t) \geqslant \omega$, $\Omega = \{\omega > 0 \text{ with } \exists k \in \mathbb{N} : \omega = 1 - k\Delta_\omega\}$ is a set of threshold values and Σ is the set of similarity measures implemented by the framework at hand. Note that Δ_ω is the first of the three parameters required by EUCLID.

In a second step, we compute the actual link specification. Given two parameters $\gamma > 1$ and $\theta > 0$, ω_i^{min} and ω_i^{max} are set to 0 and 1 respectively for each of the similarities σ_i. Then the interval ω_i^{min} and ω_i^{max} is split into γ intervals of the same size $\Delta = (|\omega_i^{max} - \omega_i^{min}|/\gamma$. For each of the possible parametrization $\omega_i = \omega_i^{min} + k\Delta$, $k \in \{0, \ldots, \gamma\}$, EUCLID simply runs all of the resulting classifiers C and compute their fitness $\mathcal{F}(C)$. The current overall best classifier $C = (\zeta_1, \ldots, \zeta_n)$ is used as new reference point. ω^i_{min} is set to $\max\{0, \zeta_i - \Delta\}$ and ω^i_{max} to $\min\{\zeta_i + \Delta, 1\}$ while $\gamma := \gamma/2$. This procedure is repeated θ times and the best overall classifier w.r.t. $\mathcal{F}$ is returned.

14.3 PSEUDO-F-MEASURES

We considered two different PFMs for the automatic discovery of link specifications. The first PFM, dubbed $\mathcal{F}_u^\beta$, was proposed by (Nikolov et al., 2012) and is based on the F_β measure. Consequently, it is defined as

$$\mathcal{F}_u^\beta = (1+\beta^2)\frac{\mathcal{P}_u \mathcal{R}_u}{\beta^2 \mathcal{P}_u + \mathcal{R}_u}. \tag{14.1}$$

Let $M \subseteq S \times T$ be a mapping generated by an algorithm, S be the set of source instances and T be the set of target instances. $\mathcal{P}_u$ is defined as

$$\mathcal{P}_u(M) = \frac{|\{s|\exists t : (s,t) \in M\}|}{\sum_s |\{t : (s,t) \in M\}|}, \tag{14.2}$$

while $\mathcal{R}_u$ is computed as follows:

$$\mathcal{R}_u(M) = \frac{|M|}{\min(|S|,|T|)}. \tag{14.3}$$

Note that $\mathcal{R}_u(M)$ can be larger than 1 as $|M|$ can be larger than $\min(|S|,|T|)$. While this does not occur in the setup proposed by (Nikolov et al., 2012), it seems rather counter-intuitive that a recall measure can lead to values beyond 1. We thus specified the following novel pseudo-recall dubbed $\mathcal{R}_d$:

$$\mathcal{R}_d(M) = \frac{|\{s|\exists t : (s,t) \in M\}| + |\{t|\exists s : (s,t) \in M\}|}{|S|+|T|}. \tag{14.4}$$

This pseudo-recall simply computes the number of 1-to-1 mappings that are covered by the mapping we generate. We would argue that it is therewith more in line with the original definition of precision and recall. Our pseudo-F-measure $\mathcal{F}_d$ is thus defined as

$$\mathcal{F}_d^\beta(M) = (1+\beta^2)\frac{\mathcal{P}_d(M)\mathcal{R}_d(M)}{\beta^2 \mathcal{P}_d(M) + \mathcal{R}_d(M)} \text{ with } \mathcal{P}_d = \mathcal{P}_u. \tag{14.5}$$

14.4 EXPERIMENTS AND RESULTS

The goal of our experiments was twofold. First, we wanted to know how deterministic approaches perform in comparison to non-deterministic approaches for the discovery of link specifications. The basic intuition here was that if deterministic approaches can achieve F-scores similar to those of non-deterministic approaches, they should be preferred as they are usually more time-efficient. We thus compared the maximal F-measure achieved by each of our approaches on the six different datasets at hand. Moreover, we wanted to measure how well PFM can predict the real performance of classifiers. Especially, we were interested in knowing whether the predictive power of pseudo-F-measures is as reliable on real data as it has been shown to be on synthetic data. Within this context, we were also interested in knowing which setting of β led to the best real F-measure across the different datasets that we used, as $\beta = 0.1$was suggested in the past (Nikolov et al., 2012). We thus ran our evaluation using both $\mathcal{F}_u$ and $\mathcal{F}_d$ for β-values between 0.1 and 2.0 using a 0.1 increment. We used two different measures to evaluate the correlation between PFM and F_1. In the following, we present the data, algorithmic parameters and correlation measures used for our experiments. We then present our results and discuss their implications for the next steps of research on link discovery.

14.4.1 *Experimental Setup*

In all experiments, we assumed that we knew the perfect mapping between the properties. Each experiment was ran on a single thread of an Ubuntu Linux server running JDK1.7. The processors were 2.0GHz quadcore Opterons. Each experiment was allocated maximally 2GB of RAM.

14.4.1.1 *Data*

We ran our experiments on three synthetic and three real datasets. The synthetic datasets consisted of widely used and well-known Persons1, Persons2 and Restaurant datasets from the OAEI2010 set of benchmark datasets. The real datasets consisted of the ACM-DBLP, Amazon-Google and Abt-Buy datasets that were extracted from websites or databases and for which a gold standard was created manually as reported in (Köpcke et al., 2009). The ACM-DBLP dataset consists of 2,617 source and 2,295 target publications (gold standard: 2,224 links). The Amazon-Google dataset links 1,363 to 3,226 products (gold standard: 1,300 links). Finally, the Abt-Buy dataset links 1,081 to 1,092 products via 1,097 correct links. All non-RDF datasets were transformed into RDF and all attribute values were set to lower case. Apart from this preprocessing, no other preprocessing step was carried out.

14.4.1.2 *Parametrization of the algorithms*

Four parameters need to be set to run EAGLE: the number of iterations, the size of the population, the crossover rate and the mutation rate. Similarly to (Nikolov et al., 2012), we used 20 iterations with a population of 100 individuals. The crossover and mutation rates were set to 0.6. Given that this approach is not deterministic, we ran the experiments 5 times and present the average values in Section 14.4.2. Note that the standard deviations for the F-measures were always under 5% of the average value. For EUCLID, we used $\Delta_\omega = 0.1$, $\gamma = 4$, $\theta = 10$.

14.4.1.3 *Correlation Measures*

To measure the accuracy of the algorithms, we used the standard F_1-measure. We were interested in determining whether the pseudo-F-measures $\mathcal{F}_u^\beta$ and $\mathcal{F}_d^\beta$ can be used practically to predict the F_1 measure achieved by an algorithm and which setting of β was the best to achieve this goal. Thus, we measured the correlation of $\mathcal{F}_u^\beta$ and $\mathcal{F}_d^\beta$ with F_1 across different values of β. We used two different correlation measures: The Pearson and the Spearman correlation. We used the *Pearson* correlation to ensure the comparability of our approach with other correlation studies as this correlation is one of the most commonly used. The main drawback of the Pearson correlation is that it is most reliable at detecting linear correlations between distributions. As there was no reason for assuming a linear relationship between our measures, we opted to also use another correlation measure that do not make any assumption upon the type of correlation between the input distributions. We used the *Spearman correlation* (Spearman, 1904), which assesses how well a monotonic function can describe the relationship between the input distributions by comparing the ranks of the values in the two input distribution. For both correlations, we used a 2-tailed significance test with a confidence threshold of 95%.

14.4.2 *Results*

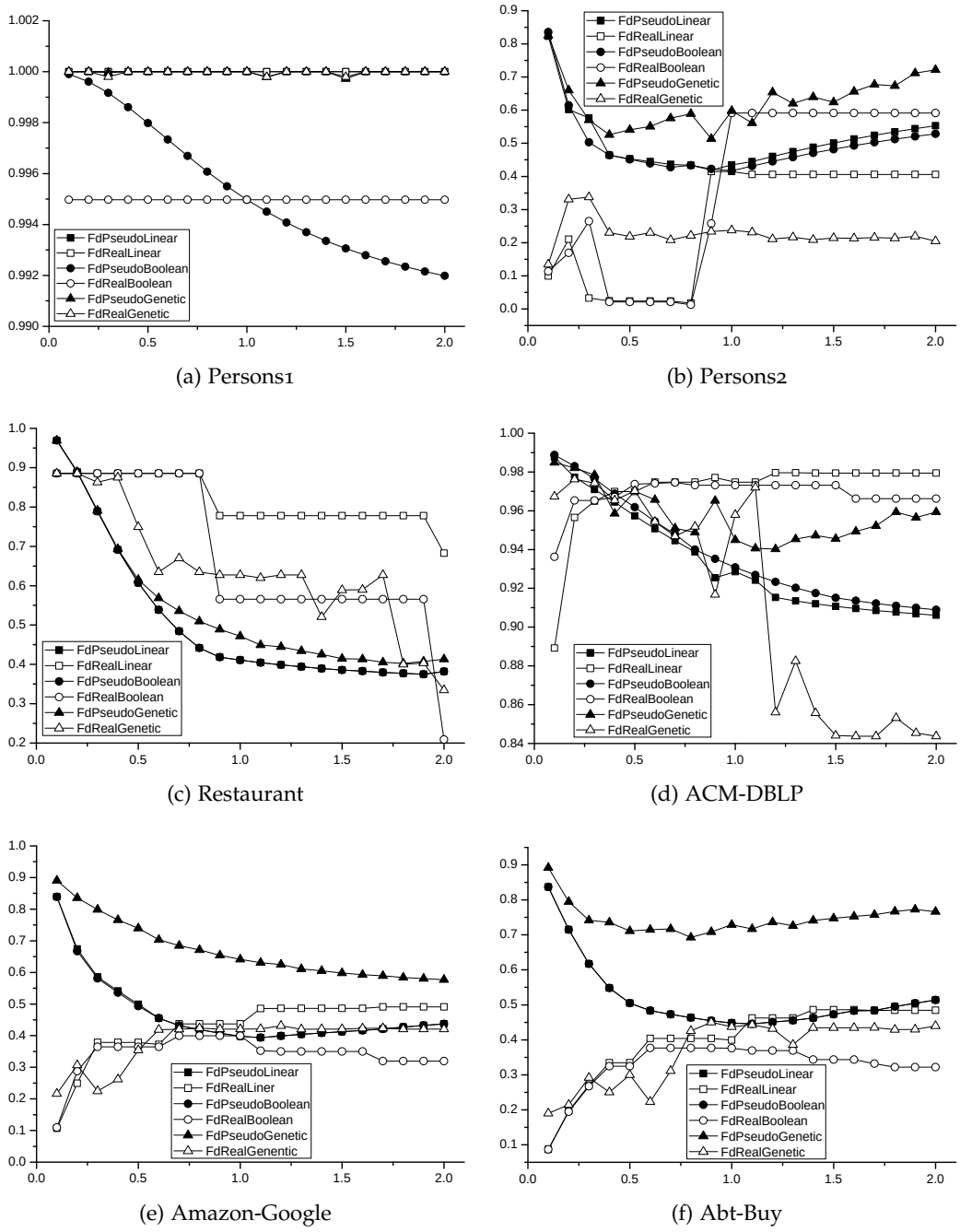
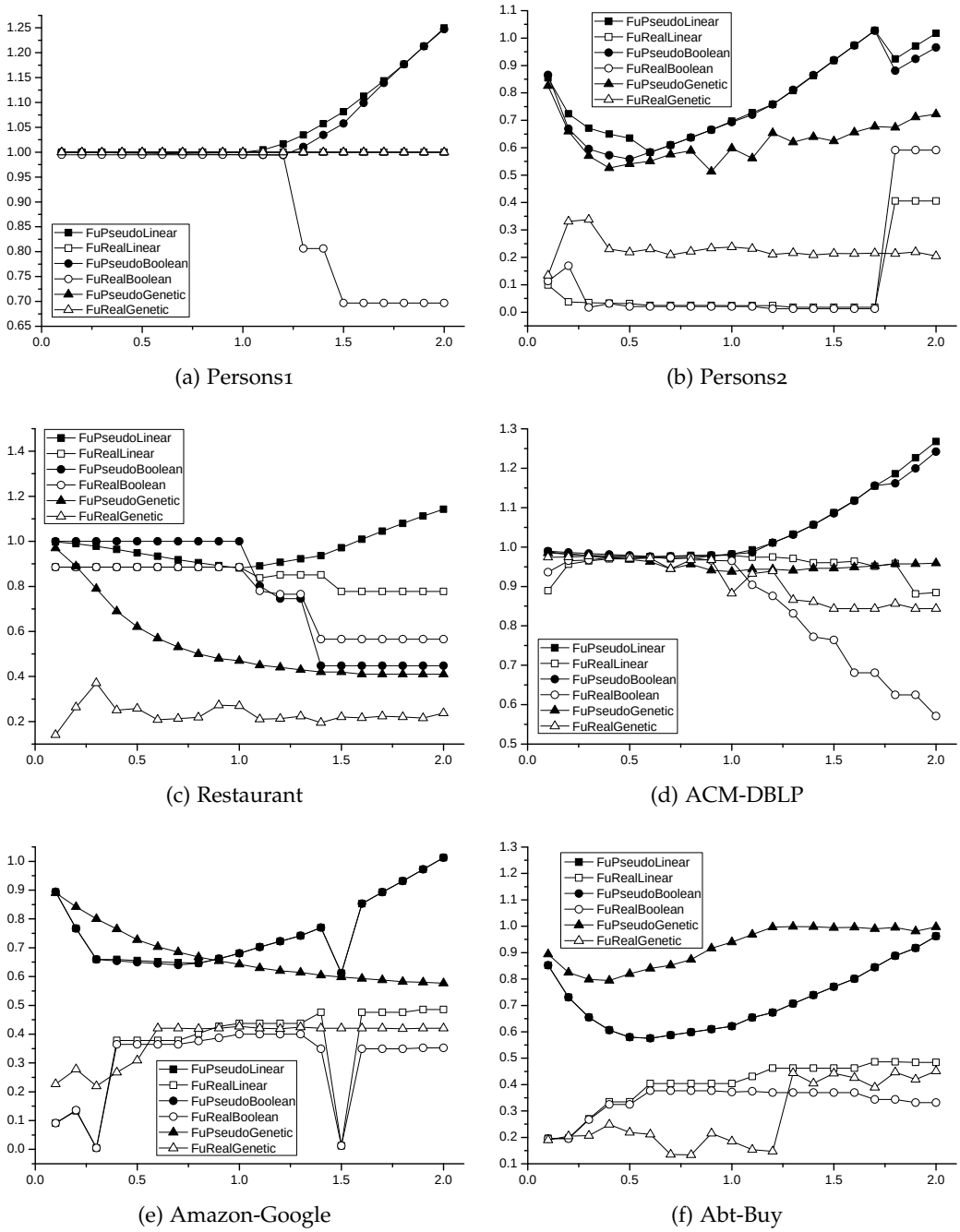


Figure 14.1: Evaluation of algorithms based on $\mathcal{F}_d$. The y-axis shows the different F-measures while the x-axis stands for different β-values. Note that "FdPseudo" stands for the pseudo-F-measures achieved by the different classifiers while "FdReal" stands for the real F-measures.

We first measured the F_1-scores achieved by our approaches when relying on $\mathcal{F}_d$ (see Figure 14.1) and $\mathcal{F}_u$ (see Figure 14.2). Our results indicate that EUCLID is in general slightly superior to EAGLE. Note that the linear classifier in combination with $\mathcal{F}_d$ even outperforms the supervised approaches presented in (Köpcke et al.,



(a) Persons1 (b) Persons2

(c) Restaurant (d) ACM-DBLP

(e) Amazon-Google (f) Abt-Buy

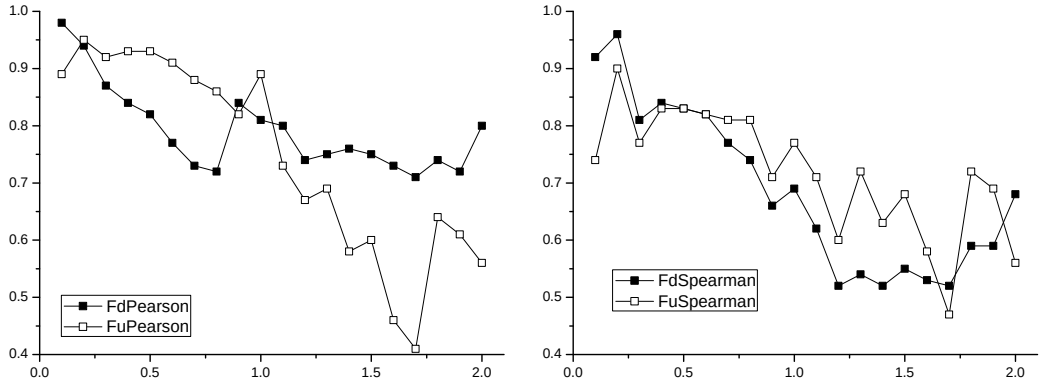
Figure 14.2: Evaluation of algorithm based on $\mathcal{F}_u$. The y-axis is the F-measure while the x-axis stands for different β-values. Note that "FuPseudo" stands for the pseudo-F-measures achieved by the different classifiers while "FuReal" stands for the real F-measures.

2009) and (Ngonga Ngomo and Lyko, 2012) on the ACM-DBLP dataset. In addition the linear model leads to better results than the Boolean model in most cases (expect on the Restaurant dataset). Our results suggest that $\mathcal{F}_d$ is better suited for EUCLID while $\mathcal{F}_u$ and $\mathcal{F}_d$ tie for EAGLE. With respect to runtime, EUCLID requires between 2.5 and 30s and is therewith between 1 and 2 orders of magnitude faster than EAGLE. Given the significant difference in runtimes we observed within our

experiments, we suggest that the development of specific algorithms for classifiers of a given type can lead to algorithms for the discovery of link specifications that are both time-efficient and highly accurate. The insight we gain is thus a clear *answer to our first question*: unsupervised deterministic approaches can perform as well as unsupervised approaches on both synthetic and real data.

	Linear		Boolean		Genetic	
	$\mathcal{F}_d$	$\mathcal{F}_u$	$\mathcal{F}_d$	$\mathcal{F}_u$	$\mathcal{F}_d$	$\mathcal{F}_u$
Persons1	**100**	**100**	99.50	99.50	**100**	**100**
Persons2	41.45	40.60	**59.12**	**59.12**	33.77	37.04
Restaurant	**88.56**	**88.56**	**88.56**	**88.56**	**88.56**	**88.56**
ACM-DBLP	**97.96**	97.94	97.46	97.46	97.62	97.71
Amazon-Google	**49.08**	48.55	39.97	39.97	43.11	42.68
Abt-Buy	**48.60**	**48.60**	37.66	37.66	45.03	45.08

Table 14.1: Maximal F-scores (in %) achieved by the approaches.



(a) Pearson (b) Spearman

Figure 14.3: Spearman and Pearson correlation of $\mathcal{F}_u^\beta$ and $\mathcal{F}_d^\beta$ across different values of β measured independently from the algorithm used.

To answer our second question, we first computed the algorithm-independent correlation of $\mathcal{F}_d$ and the F_1 measure as well as the correlation of $\mathcal{F}_u$ and the F_1 measure (see Figure 14.3). In our experiments, the correlations varied significantly across the different values of β, yet remained positive and significant in 97.5% of the cases (the 2 lowest correlation scores of the Pearson correlation for $\mathcal{F}_u$ were not significant). This means that optimizing $\mathcal{F}_u$ and $\mathcal{F}_d$ for one particular setting of β is a sensible approach towards finding the best classifier for that particular setting of β. We then computed the Pearson and Spearman correlation (see Table 14.2) between $\mathcal{F}_u$, $\mathcal{F}_d$ and the F_1 measure achieved by the different approaches across different values of β. Our results were somewhat surprising as we detected both significant positive and negative correlations across the different datasets and for both correlations. Interestingly, while the number of significant positive and negative correlations were relatively balanced for the synthetic datasets, negative correlations seemed to dominate the set of real datasets, thus hinting towards $\mathcal{F}_u$

	Linear		Boolean		Genetic	
	$\mathcal{F}_d$	$\mathcal{F}_u$	$\mathcal{F}_d$	$\mathcal{F}_u$	$\mathcal{F}_d$	$\mathcal{F}_u$
Persons1	–	–	–	-0.85*	0.84*	-0.1
Persons2	-0.09	0.54*	-0.12	0.43	-0.43	-0.43
Restaurant	0.73*	-0.71*	0.71*	1*	0.84*	0.16
ACM-DBLP	-0.70*	-0.65*	-0.43	-0.97*	0.47*	0.51*
Amazon-Google	-0.95*	0.49	-0.79*	0.07	-0.88*	-0.03
Abt-Buy	-0.9*	0.27	-0.98*	-0.27	0.38	-0.9*
Persons1	–	–	–	-0.87*	0.99*	-0.1
Persons2	0.02	-0.09	0.21	-0.11	-0.56*	-0.56*
Restaurant	0.85*	-0.54*	0.85*	1*	0.91*	0.15
ACM-DBLP	-0.87*	-0.73*	0.05	-0.95*	0.34	0.47*
Amazon-Google	-0.49*	0.68*	-0.27	-0.22	-0.64*	-0.39
Abt-Buy	-0.37	0.59*	-0.82*	-0.47*	-0.13	-0.49*

Table 14.2: Pearson and Spearman Correlation of PFM and real F-measures across different β-values. The top section of the table shows the Pearson correlation while the bottom part shows the Spearman correlation. The correlations are not defined for the fields marked with "–" due to at least one of the standard deviations involved being 0.

and $\mathcal{F}_d$ behaving differently depending on the type of data they are confronted with. The negative correlation values suggest that to detect the best values of β for a real dataset automatically, the mapping M which leads to the smallest best value of $\mathcal{F}_d$ across the different values of β should be chosen. This seems rather counter-intuitive and is a hypothesis that requires ampler testing on a larger number of real datasets. Overall, our results show clearly that no β-value achieves a maximal F_1-measure across our datasets. Still, for real datasets, $\mathcal{F}_d$ seems to perform well for $\beta \in [0.8, 1.2]$. Stating such an interval for $\mathcal{F}_u$ is more difficult as the set of β-values that lead to the best mapping is very heterogeneous across the different datasets. Interestingly, this conclusion diverges from that proposed in previous work. The answer to our second question is still clearly that while the predictive power of $\mathcal{F}_u^\beta$ and $\mathcal{F}_d^\beta$ is sufficient for the results to be used in practical settings, significant effort still needs to investigated to create a generic non-parametric PFM that can be used across different datasets and algorithms to predict the F_1-measure reliably .

14.5 CONCLUSION

In this chapter, we present a first series of experiments to determine how well standard classifier models such as linear and Boolean classifiers perform in comparison to classifiers generated by the means of genetic programming in an unsupervised learning setting based on maximizing a PFM. Overall, our results indicate that we are still at the beginning of the search towards the "holy grail" of PFMs. Especially on real data, the maximal PFM achieved by algorithms across different

values of β is often negatively correlated with the value of the F_1. The magnitude of this effect is significantly reduced on synthetic data. This difference suggest that there is still a need for benchmark generation methods that allow creating benchmark datasets which reflect real data in a more holistic way. Moreover, our evaluation shows that deterministic classifiers perform as well as or better than non-deterministic approaches while still bearing the main advantage of being significantly more time-efficient. Thus, finding more efficient extension of EUCLID or similar approaches should allow providing users of LD frameworks with accurate link specifications within an interactive setting. Detecting the right parametrization for PFM yet remains an unsolved problem.

15

UNSUPERVISED LINK DISCOVERY THROUGH KNOWLEDGE BASE REPAIR

PREAMBLE

This chapter presents COLIBRI, an iterative unsupervised approach for link discovery and is taken from (Ngonga Ngomo et al., 2014). COLIBRI allows the discovery of links between n datasets ($n \geqslant 2$) while improving the quality of the instance data in these datasets. To this end, COLIBRI combines error detection and correction with unsupervised link discovery. The ideas and algorithms in this paper were developed and implemented by the author, who also supervised the evaluation and co-wrote the paper.

15.1 INTRODUCTION

Over the last years, the Linked Open Data cloud has evolved from a mere 12 to more than 300 knowledge bases (Auer et al., 2013a). The basic architectural principles behind this data compendium are akin to those of the document Web and thus decentralized in nature.[1] This architectural choice has led to knowledge pertaining to the same domain being published by independent entities in the Linked Open Data cloud. For example, information on drugs can be found in Diseasome[2] as well as DBpedia[3] and Drugbank.[4] Moreover, certain datasets such as DBLP have been published by several bodies,[5] leading to duplicated content in the Data Web. With the growth of the number of independent data providers, the concurrent publication of datasets containing related information promises to become a phenomenon of increasing importance. Enabling the joint use of these datasets for tasks such as federated queries, cross-ontology question answering and data integration is most commonly tackled by creating links between the resources described in the datasets. Devising accurate link specifications (also called linkage rules (Isele and Bizer, 2011)) to compute these links has yet been shown to be a difficult and time-consuming problem in previous works (Isele and Bizer, 2011; Isele et al., 2011; Ngonga Ngomo et al., 2013; Nikolov et al., 2012).

The insight behind this work is that declarative link specifications (e.g., SILK and LIMES specifications) compare the property values of resources by using similarity functions to determine whether they should be linked. For example, imagine being given three knowledge bases K_1 that contains cities, K_2 that contains provinces and K_3 that contains countries as well as the `dbo:locatedIn` predicate[6] as relation. The specification that links K_1 to K_2 might compare province labels while the specifications that link K_1 and K_2 to K_3 might compare country labels. Imagine the

1 See http://www.w3.org/DesignIssues/LinkedData.html.
2 http://wifo5-03.informatik.uni-mannheim.de/diseasome/
3 http://dbpedia.org
4 http://wifo5-03.informatik.uni-mannheim.de/drugbank/
5 http://dblp.l3s.de/, http://datahub.io/dataset/fu-berlin-dblp and http://dblp.rkbexplorer.com/.
6 The prefix `dbo:` stands for `http://dbpedia.org/ontology/`.

city `Leipzig` in K_1 were linked to `Saxony` in K_2 and to `Germany` in K_3. In addition, imagine that `Saxony` were erroneously linked to `Prussia`. If we assume the first Linked Data principle (i.e., "Use URIs as names for things")[7], then the following holds: By virtue of the transitivity of `dbo:locatedIn` and of knowing that it is a many-to-1 relation,[8] we can deduce that one of the links in this constellation must be wrong. Note that this inference would hold both under open- and closed-world assumptions. Thus, if we knew the links between `Leipzig` and `Germany` as well as `Leipzig` and `Saxony` to be right, we could then repair the value of the properties of `Saxony` that led it to be linked to `Prussia` instead of `Germany` and therewith ensure that is linked correctly in subsequent link discovery processes.

We implement this intuition by presenting COLIBRI, a novel iterative and unsupervised approach for LD. COLIBRI uses link discovery results for *transitive many-to-1 relations* (e.g., `locatedIn` and `descendantSpeciesOf`) and *transitive 1-to-1 relations* (e.g., `owl:sameAs`) between instances in knowledge bases for the sake fo attempting to repair the instance knowledge in these knowledge bases and improve the overall quality of the links. In contrast to most of the current unsupervised LD approaches, COLIBRI takes an n-set[9] of set of resources $K_1, \ldots, K_n$ with $n \geqslant 2$ as input. In a *first step*, our approach applies an *unsupervised machine-learning* approach to each pair (K_i, K_j) of sets of resources (with $i \neq j$). By these means, COLIBRI generates $n(n-1)$ mappings. Current unsupervised approaches for LD would terminate after this step and would not make use of the information contained in some mappings to improve other mappings. The intuition behind COLIBRI is that using such information can help improve the overall accuracy of a link discovery process if the links are *many-to-1 and transitive* or *1-to-1 and transitive*. To implement this insight, all mappings resulting from the first step are forwarded to a *voting approach* in a *second step*. The goal of the voting approach is to detect possible errors within the mappings that were computed in the previous step (e.g., missing links). This information is subsequently used in the *third step* of COLIBRI, which is the *repair step*. Here, COLIBRI first detects the sources of errors in the mappings. These sources of errors can be wrong or missing property values of the instances. Once these sources of errors have been eliminated, a new iteration is started. COLIBRI iterates until a termination condition (e.g., a fixpoint of its objective function) is met.

Overall, the main contributions of this work are as follows:

- We present the (to the best of our knowledge) the first unsupervised LD approach that attempts to repair instance data for improving the link discovery process.
- Our approach is the first unsupervised LD approach that can be applied to $n \geqslant 2$ knowledge bases and which makes use of the intrinsic topology of the Web of Data.
- We evaluate our approach on six datasets. Our evaluation shows that we can improve the results of state-of-the-art approaches w.r.t. the F-measure while reliably detecting and correcting errors in instance data.

7 http://www.w3.org/DesignIssues/LinkedData.html

8 From this characteristic, we can infer that (1) a city cannot be located in two different provinces, (2) a city cannot be located in two different countries and (3) a province cannot be located in two different countries.

9 An n-set is a set of magnitude n.

We rely on EUCLID (Ngonga Ngomo and Lyko, 2013) as machine-learning approach and thus provide a fully deterministic approach. We chose EUCLID because it performs as well as non-deterministic approaches on the datasets used in our evaluation (Ngonga Ngomo and Lyko, 2013) while presenting the obvious advantage of always returning the same result for a given input and a given setting. Moreover, it is not tuned towards discovery exclusively `owl:sameAs` links (Suchanek et al., 2011). Still, COLIBRI is independent of EUCLID and can be combined with any link specification learning approach. The approaches presented herein were implemented in LIMES.[10]

15.2 PRELIMINARIES

In this section, we present some of the notation and concepts necessary to understand the rest of the paper. We use Figure 15.1 to exemplify our notation. The formalization of LD provided below is an extension of the formalization for 2 input knowledge bases presented in (Ngonga Ngomo, 2012b). Given n knowledge bases $K_1 \dots K_n$, LD aims to discover pairs $(s_i, s_j) \in K_i \times K_j$ that are such that a given relation $\mathcal{R}$ holds between s_i and s_j. The direct computation of the pairs for which $\mathcal{R}$ holds is commonly very tedious if at all possible. Thus, most frameworks for LD resort to approximating the set of pairs for which $\mathcal{R}$ holds by using *link specifications* (LS). A LS can be regarded as a classifier C_{ij} that maps each element of the Cartesian product $K_i \times K_j$ to one of the classes of $Y = \{+1, -1\}$, where K_i is called the *set of source instances* while K_j is the *set of target instances*. $(s, t) \in K_i \times K_j$ is considered by C_{ij} to be a correct link when $C_{ij}(s, t) = +1$. Otherwise, (s, t) is considered not to be a potential link. In our example, C_{12} returns $+1$ for $s =$ `ex1:JohnDoe` and $t =$ `ex2:JD`.

We will assume that the classifier C_{ij} relies on comparing the value of complex similarity function $\sigma_{ij} : K_i \times K_j \to [0, 1]$ with a threshold θ_{ij}. If $\sigma_{ij}(s, t) \geqslant \theta_{ij}$, then the classifier returns $+1$ for the pair (s, t). In all other cases, it returns -1. The complex similarity function σ_{ij} consists of a combination of atomic similarity measures $\pi^l_{ij} : K_i \times K_j \to [0, 1]$. These atomic measures compare the value of a particular property of $s \in K_i$ (for example its `rdfs:label`) with the value of a particular property of $t \in K_j$ (for example its `:name`) and return a similarity score between 0 and 1. In our example, σ_{12} relies on the single atomic similarity function $\mathtt{trigrams}(\mathtt{:ssn}, \mathtt{:ssn})$, which compares the social security number attributed to resources of K_1 and K_2.

We call the set of all pairs $(s, t) \in K_i \times K_j$ that are considered to be valid links by C_{ij} a *mapping*. We will assume that the resources in each of the knowledge bases $K_1, \dots, K_n$ can be ordered (e.g., by using the lexical ordering of their URI) and thus assigned an index. Then, a mapping between the knowledge bases K_i and K_j can be represented as a matrix M_{ij} of dimensions $|K_i| \times |K_j|$, where the entry in the x^{th} row and y^{th} column is denoted $M_{ij}(x, y)$. If the classifier maps (s, t) to -1, then $M_{ij}(x, y) = 0$ (where x is the index of s and y is the index of t). In all other cases, $M_{ij}(x, y) = \sigma(s, t)$. For the sake of understandability, we will sometimes write $M_{ij}(s_x, t_y)$ to signify $M_{ij}(x, y)$. In our example, C_{34} is a linear classifier, $\sigma_{34} = \mathtt{trigrams}(\mathtt{:id}, \mathtt{:id})$ and $\theta_{34} = 1$. Thus, (`ex3:J36`, `ex4:Cat40_1`) is considered a link.

10 http://limes.sf.net

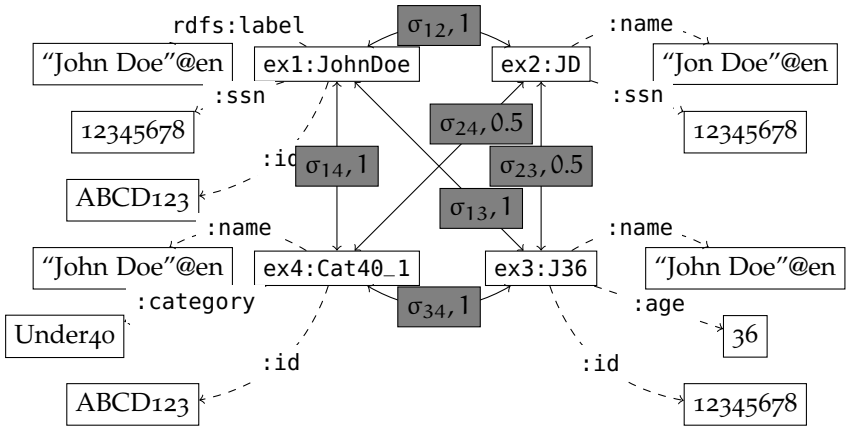


Figure 15.1: Example of four linked resources from four different knowledge bases. The white nodes are resources or literals. Properties are represented by dashed labeled arrows. Links are represented by plain arrows. The gray boxes on the links show the names of the similarity measures used to link the resources they connect as well as the similarity value for each of these resource pairs. $\sigma_{12} = \mathit{trigrams}(\texttt{:ssn}, \texttt{:ssn})$, $\sigma_{13} = \sigma_{14} = \mathit{trigrams}(\texttt{:id}, \texttt{:id})$, $\sigma_{23} = \sigma_{24} = \sigma_{34} = \mathit{dice}(\texttt{:name}, \texttt{:name})$, $\sigma_{ij} = \sigma_{ji}$.

Supervised approaches to the computation of link specifications use labelled training data $L \subseteq K_i \times K_j \times Y$ to minimize the error rate of C_{ij}. Colibri relies on an unsupervised approach. The idea behind *unsupervised approaches* to learning link specifications is to refrain from using any training data (i.e., $L = \emptyset$). Instead, unsupervised approaches aim to optimize an *objective function*. The objective functions we consider herein approximate the value of the F-measure achieved by a specification and are thus dubbed pseudo-F-measures (short: PFMs) (Nikolov et al., 2012).

In this work, we extend the PFM definition presented in (Ngonga Ngomo and Lyko, 2013). Like in (Nikolov et al., 2012; Suchanek et al., 2011; Hassanzadeh et al., 2013), the basic assumption behind this PFM is that one-to-one links exist between the resources in S and T. We chose to extend this measure to ensure that it is symmetrical w.r.t. to the source and target datasets, i.e., PFM(S, T) = PFM(T, S). Our pseudo-precision $\mathcal{P}$ computes the fraction of links that stand for one-to-one links and is equivalent to the strength function presented in (Hassanzadeh et al., 2013). Let $\mathit{links}(K_i, M_{ij})$ be the subset of K_i whose elements are linked to at least one element of K_j. Then, $\mathcal{P}(M_{ij}) = \frac{|\mathit{links}(K_i, M_{ij})| + |\mathit{links}(K_j, M_{ij})|}{2|M_{ij}|}$. The pseudo-recall $\mathcal{R}$ computed the fraction of the total number of resources (i.e., $|K_i| + |K_j|$) from that are involved in at least one link: $\mathcal{R}(M_{ij}) = \frac{|\mathit{links}(K_i, M_{ij})| + |\mathit{links}(K_j, M_{ij})|}{|K_i| + |K_j|}$. Finally, the PFM $\mathcal{F}_\beta$, is defined as $\mathcal{F}_\beta = (1 + \beta^2)\frac{\mathcal{P}\mathcal{R}}{\beta^2\mathcal{P} + \mathcal{R}}$.

For the example in Figure 15.2, $\mathcal{P}(M_{12}) = 1$, $\mathcal{R}(M_{12}) = \frac{2}{3}$ and $\mathcal{F}_1 = \frac{4}{5}$. Our PFM works best if S and T are of comparable size and one-to-one links are to be detected. For example, Euclid achieves 99.7% F-measure on the OAEI Persons1 dataset.[11] It even reaches 97.7% F-measure on the DBLP-ACM dataset, therewith outperforming the best supervised approach (FEBRL) reported in (Köpcke et al., 2010). Yet, Euclid achieves worse results compared to FEBRL on the Amazon-Google Products dataset with an F-measure of 43% against 53.8%, where $|T| \approx 3|S|$.

11 http://oaei.ontologymatching.org/

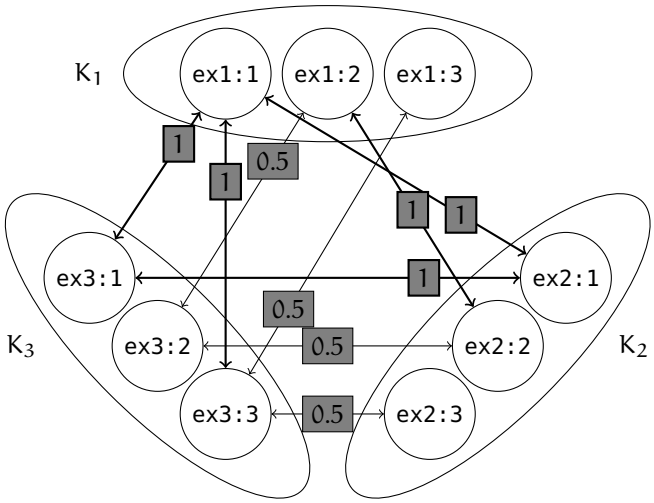


Figure 15.2: Example of mappings between 3 sets of resources. K_1 has the namespace ex1, K_2 the namespace ex2 and K_3 the namespace ex3. Thick lines stand for links with the similarity value 1 while thin lines stand for links with the similarity value 0.5.

15.3 THE COLIBRI APPROACH

In this section, we present the COLIBRI approach and its components in detail. We begin by giving an overview of the approach. Then, for the sake of completeness, we briefly present EUCLID, the unsupervised LD approach currently underlying COLIBRI. For more information about EUCLID, please see (Ngonga Ngomo and Lyko, 2013). Note that COLIBRI can be combined with any unsupervised LD approach. After the overview of EUCLID, we present the voting approach with which COLIBRI attempts to detect erroneous or missing links. In a final step, we present how COLIBRI attempts to repair these sources of error.

15.3.1 *Overview*

Most of the state-of-the-art approaches to LD assume scenarios where two sets of resources are to be linked. COLIBRI assumes that it is given n sets of resources $K_1, \ldots, K_n$. The approach begins by computing mappings M_{ij} between resources of pairs of sets of resources (K_i, K_j). To achieve this goal, it employs the EUCLID algorithm (Ngonga Ngomo and Lyko, 2013) described in the subsequent section. The approach then makes use of the transitivity of $\mathcal{R}$ by computing voting matrices V_{ij} that allow detecting erroneous as well as missing links. This information is finally used to detect resources that should be repaired. An overview of COLIBRI is given in Algorithm 15.1. In the following sections, we explain each step of the approach.

15.3.2 *EUCLID*

Over the last years, non-deterministic approaches have been commonly used to detect highly accurate link specifications (see, e.g., (Ngonga Ngomo et al., 2013; Nikolov et al., 2012)). EUCLID (Line 8 of Algorithm 15.1) is a deterministic unsu-

Algorithm 15.1 The COLIBRI approach. $\mathcal{M}$ stands for the set of all M_{ij} while $\tilde{\mathcal{V}}$ stands for the set of all $\tilde{V}_{ij}$. The $maxIterations$ parameter ensures that the approach terminates.

```
1: F_new := 0, F_old := 0, iterations = 0
2: while F_new − F_old > 0 do
3:    F_old := F_new
4:    F_new := 0
5:    for i ∈ {1, ..., n} do
6:       for j ∈ {1, ..., n}, j ≠ i do
7:          M_ij := EUCLID (K_i, K_j)
8:          F_new := F_new+PSEUDOF(M_ij)
9:       end for
10:   end for
11:   F_new := F_new/(n(n − 1))
12:   if F_new − F_old > 0 then
13:      for i ∈ {1, ..., n} do
14:         for j ∈ {1, ..., n}, j ≠ i do
15:            V_ij :=COMPUTEVOTING(M_ij, M)
16:            Ṽ_ij =POSTPROCESS(V_ij)
17:         end for
18:      end for
19:      for (a, b) ∈ GETWORSTLINKS(Ṽ) do
20:         (rs, rt) :=GETREASON(a, b)
21:         REPAIR(rs,rt)
22:      end for
23:   end if
24: end while
```

pervised approach for learning link specifications. The core idea underlying the approach is that link specifications of a given type (linear, conjunctive, disjunctive) can be regarded as points in a link specification space. Finding an accurate link specification is thus equivalent to searching through portions of this specification space. In the following, we will assume that EUCLID tries to learn a conjunctive classifier, i.e., a classifier which returns $+1$ for a pair $(s,t) \in K_i \times K_j$ when $\bigwedge_{l=1}^{m} (\pi_{ij}^l(s,t) \geqslant \theta_{ij}^l)$ holds. The same approach can be used to detect disjunctive and linear classifiers. EUCLID assumes that it is given a set of m atomic similarity functions π_{ij}^l with which it can compare $(s,t) \in K_i \times K_j$. The atomic functions π_{ij}^l build the basis of an m-dimensional space where each of the dimensions corresponds to exactly one of the π_{ij}^l. In this space, the specification $\bigwedge_{l=1}^{m} (\pi_{ij}^l(s,t) \geqslant \theta_{ij}^l)$ has the coordinates $(\theta_{ij}^1, \ldots, \theta_{ij}^m)$. The core of EUCLID consists of a hierarchical grid search approach that aims to detect a link specification within a hypercube (short: cube) which maximizes the value of a given objective function $\mathcal{F}$. The hypercubes considered by EUCLID are such that their sides are all orthogonal to the axes of the space. Note that such a hypercube can be described entirely by two points $b = (b_1, \ldots, b_m)$ and $B = (B_1, \ldots, B_m)$ with $\forall i \in \{1, \ldots, m\}(b_i \leqslant B_i)$.

EUCLID begins by searching through the cube defined by $b = \underbrace{(0,\ldots,0)}_{m \text{ times}}$ and $B = \underbrace{(1,\ldots,1)}_{m \text{ times}}$ (i.e., the whole of the similarity space). A point w with coordinates $(w_1,\ldots,w_m)$ corresponds to the classifier with the specific function $\bigwedge_{l=1}^{m}(\pi_{ij}^{l}(s_i,s_j) \geqslant w_l)$. Let $\alpha \in \mathbb{N}, \alpha \geqslant 2$ be the granularity parameter of EUCLID. The search is carried out by generating a grid of $(\alpha+1)^m$ points g whose coordinates $g_i = \left(b_i + k_i\frac{(B_i-b_i)}{\alpha}\right)$, where $k_i \in \{0,\ldots,\alpha\}$. We call $\Delta_i = \frac{(B_i-b_i)}{\alpha}$ the *width* of the grid in the ith dimension. EUCLID now computes the pseudo-F-measure $\mathcal{F}$ of the specification corresponding to each point on the grid. Let g^{max} be a point that maximizes $\mathcal{F}$. Then, EUCLID updates the search cube by updating the coordinates of the points b and B as follows: $b_i = \left(\max\left\{0, g_i^{max} - \Delta_i\right\}\right)$ and $B_i = \left(\min\left\{1, g_i^{max} + \Delta_i\right\}\right)$. Therewith, EUCLID defines a new and smaller search cube. The search is iterated until a stopping condition such as a given number of iterations is met.

15.3.3 *Voting*

The result of EUCLID is a set of $n(n-1)$ mappings M_{ij} which link the resource set K_i with the resource set K_j. The goal of the second step of a COLIBRI iteration is to determine the set of resources that might contain incomplete or erroneous information based on these mappings. The basic intuition behind the approach is to exploit the transitivity of the relation $\mathcal{R}$ is as follows: If the link $(s,t) \in K_i \times K_j$ is correct, then for all k with $1 \leqslant k \leqslant n$ with $k \neq i,j$, there should exist pairs of links (s,z) and (z,t) with $M_{ik}(s,z) > 0$ and $M_{kj}(z,t) > 0$. Should such pairs not exist or be weakly connected, then we can assume that some form of error was discovered.

Formally, we go about implementing this intuition as follows: We first define the voting matrices V_{ij} as $V_{ij} = \frac{1}{n}\left(M_{ij} + \sum_{k=0,k\neq i,j}^{n} M_{ik}M_{kj}\right)$ (Line 15 of Algorithm 15.1). In the example shown in Figure 15.2, the mappings are

$$M_{12} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}, M_{13} = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 0.5 & 0 \\ 0 & 0 & 0.5 \end{pmatrix} \text{ and } M_{23} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0 & 0.5 \end{pmatrix}.$$

The corresponding voting matrices are thus

$$V_{12} = \begin{pmatrix} 1 & 0 & 0.25 \\ 0 & 0.625 & 0 \\ 0 & 0 & 0.125 \end{pmatrix}, V_{13} = \begin{pmatrix} 1 & 0 & 0.5 \\ 0 & 0.5 & 0 \\ 0 & 0 & 0.25 \end{pmatrix} \text{ and } V_{23} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0 & 0.25 \end{pmatrix}.$$

Each voting matrix V_{ij} encompasses the cumulative results of the linking between all pairs of resource sets with respect to the resources in (K_i, K_j). Computing V_{ij} as given above can lead to an explosion in the number of resources associated to s_i. In our example, the erroneous link between `ex1:1` and `ex3:3` leads to `ex1:1` being linked not only to `ex2:1` but also to `ex2:3` in V_{12}. We thus post-process each V_{ij} by only considering the best match for each $s \in K_i$ within V_{ij}, i.e., by removing each non-maximal entry from each row of V_{ij} (Line 16 of Algorithm 15.1). We label the resulting matrix $\tilde{V}_{ij}$. For our example, we get the following matrices:

$$\tilde{V}_{12} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0.625 & 0 \\ 0 & 0 & 0.125 \end{pmatrix}, \tilde{V}_{13} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0 & 0.25 \end{pmatrix} \text{ and } \tilde{V}_{23} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0 & 0.25 \end{pmatrix}.$$

COLIBRI now assumes that the links encoded in $\tilde{V}_{ij}$ are most probably correct. All entries of $\tilde{V}_{ij}$ being 1 are thus interpreted as all matrices agreeing on how to link the resources in (K_i, K_j). In the example in Figure 15.2, this is the case for $\tilde{V}_{12}(\texttt{ex1:1}, \texttt{ex2:1})$. Should this not be the case, then the disagreement between the matrices can result from the following reasons:

1. *Missing links*: This is the case in our example for the link $(\texttt{ex1:3}, \texttt{ex2:3})$ which is not contained in M_{12}. For this reason, $\tilde{V}_{12}(\texttt{ex1:3}, \texttt{ex2:3})$ is minimal.
2. *Weak links*: This is the case for the second-lowest entry in $\tilde{V}_{12}$, where the entry for $(\texttt{ex1:2}, \texttt{ex2:2})$ is due to $M_{13}(\texttt{ex1:2}, \texttt{ex3:2})$ and $M_{32}(\texttt{ex3:2}, \texttt{ex2:2})$ being 0.5.

COLIBRI now makes use of such disagreements to repair the entries in the knowledge bases with the aim of achieving a better linking. To this end, it selects a predetermined number of links (a, b) over all $\tilde{V}_{ij}$ whose weight is minimal and smaller than 1 (GETWORSTLINKS in Algorithm 15.1). These links are forwarded to the instance repair.

15.3.4 *Instance repair*

For each of the links (a, b) selected by the voting approach, the instance repair routine of COLIBRI begins by computing why $\tilde{V}_{ij}(a, b) < 1$. To achieve this goal, it computes the *reason* $(rs, rt) \in \left(K_i \times \bigcup_{k=1,k\neq i}^{n} K_k\right) \cup \left(\bigcup_{k=1,k\neq j}^{n} K_k \times K_j\right)$ by detecting the smallest entry that went into computing $\tilde{V}_{ij}(a, b)$. Three possibilities occur:

1. $(rs, rt) \in K_i \times K_j$: In this case, the weak or missing link is due to the initial mapping M_{ij}.
2. $(rs, rt) \in K_i \times K_k$ with $k \neq i \wedge k \neq j$: In this case, the weak or missing link is due to the in-between mapping M_{ik}.
3. $(rs, rt) \in K_k \times K_j$ with $k \neq i \wedge k \neq j$: Similarly to the second case, the weak or missing link is due to the in-between mapping M_{kj}.

In all three cases, the repair approach now aims to improve the link by repairing the resource rs or rt that most probably contains erroneous or missing information. To achieve this goal, it makes use of the similarity measure σ used to generate (rs, rt). The value of this measure being low suggests that the property values p^l and q^l used across the similarity measures π^l are dissimilar. The idea of the repair is then to overwrite exclusively the values of $p^l(rs)$ with those of $q^l(rt)$ or vice-versa. The intuition behind deciding upon whether to update rs or rt is based on the *average similarity* $\bar{\sigma}(rs)$ resp. $\bar{\sigma}(rt)$ of the resources rs and rt to other resources. For a resource $s \in K_i$, this value is given by

$$\bar{\sigma}(s) = \frac{1}{n-1}\left(\sum_{k=1,k\neq i}^{n} \max_{t \in K_k} \sigma_{ik}(s, t)\right). \tag{15.1}$$

Here, the assumption is that the higher the value of $\bar{\sigma}$ for a given resource, the higher the probability that it does not contain erroneous information.

Let us consider anew the example given in Figure 15.2 and assume that the link that is to be repaired is (ex1:2, ex2:2). One reason for this link would be rs = ex1:2 and rt = ex3:2. Now $\bar{\sigma}$(ex1:2) = 0.75 while $\bar{\sigma}$(ex3:2) = 0.5. COLIBRI would thus choose to overwrite the values of ex3:2 with those of ex1:2.

The overwriting in itself is carried out by overwriting the values of $q^l(rt)$ with those of $p^l(rs)$ if $\bar{\sigma}(rs) \geqslant \bar{\sigma}(rt)$ and vice-versa. This step terminates an iteration of COLIBRI, which iterates until a termination condition is reached, such as the average value of $\mathcal{F}$ for the mappings generated by EUCLID declining or a maximal number of iterations. The overall complexity of each iteration of COLIBRI is $O(n^2 \times E)$, where E is the complexity of the unsupervised learning algorithm employed to generate the mappings. Thank to the algorithms implemented in LIMES which have a complexity close to $O(m)$ where $m = \max\{|S|, |T|\}$ for each predicate, EUCLID has a complexity of $O(pm)$, where p is the number of predicates used to compare entities. Consequently, the overall complexity of each iteration of COLIBRI is $O(pmn^2)$ when it relies on EUCLID. While we observed a quick converge of the approach on real and synthetic datasets within our evaluation (maximally 10 iterations), the convergence speed of the approach may vary on the datasets used.

15.4 EVALUATION

The aim of our evaluation was to measure whether COLIBRI can improve the F-measure of mappings generated by unsupervised link discovery approaches. To this end, we measured the increase in F-measure achieved by COLIBRI w.r.t to the number of iterations it carried out on a synthetic dataset generated out of both synthetic and real data. To the best of our knowledge, no benchmark dataset is currently available for link discovery across $n > 2$ knowledge bases. We thus followed the benchmark generation approach for instance matching presented in (Ferrara et al., 2011) to generate the evaluation data for COLIBRI.

15.4.1 *Experimental Setup*

We performed controlled experiments on data generated automatically from two synthetic and three real datasets. The synthetic datasets consisted of the Persons1 and Restaurant datasets from the OAEI2010 benchmark datasets.[12] The real datasets consisted of the ACM-DBLP, Amazon-Google and Abt-Buy datasets.[13] We ran all experiments in this section on the source dataset of each of these benchmark datasets (e.g., ACM for ACM-DBLP). We omitted OAEI2010's Person2 because its source dataset is similar to Person1's. Given the lack of benchmark data for link discovery over several sources, we generated a synthetic benchmark as follows: Given the initial source dataset K_1, we first generated $n - 1$ copies of K_1. Each copy was altered by using a subset of the operators suggested in (Ferrara et al., 2011). The alteration strategy consisted of randomly choosing a property of a randomly chosen resource and altering it. We implemented three syntactic operators to alter property values, i.e., misspellings, abbreviations and word permutations. The syntactic

12 Available online at http://oaei.ontologymatching.org/2010/.

13 Available online at http://dbs.uni-leipzig.de/en/research/projects/object_matching/fever/benchmark_datasets_for_entity_resolution.

operator used for altering a resource was chosen randomly. We call the probability of a resource being chosen for alteration the *alteration probability* (ap). The goal of this series of experiments was to quantify (1) the gain in F-measure achieved by COLIBRI over EUCLID and (2) the influence of ap and of the number n of knowledge bases on COLIBRI's F-measure.

The F-measure of EUCLID and COLIBRI was the average F-measure they achieved over all pair (K_i, K_j) with $i \neq j$. To quantify the amount of resources that were altered by COLIBRI in the knowledge bases $K_1, \ldots, K_n$, we computed the average *error rate* in the knowledge bases after each iteration as follows:

$$errorrate = 1 - \frac{1}{n(n-1)} \sum_{i=1}^{n} \sum_{j=1, j\neq i}^{n} \frac{2|K_i \cap K_j|}{|K_i| + |K_j|}. \tag{15.2}$$

The maximal number of COLIBRI iterations was set to 10. We present the average results but omit the standard deviations for the sake of legibility. For precision, the standard deviation was maximally 4%. The recall's standard deviation never exceeded 1% while it reached 2% for the F-measure.

15.4.2 *Experimental Results*

We varied the number of knowledge bases between 3 and 5. Moreover, we varied the alteration probability between 10% and 50% with 10% increments. We then measured the precision, recall, F-measure, runtime and number of repairs achieved by the batch version of COLIBRI over several iterations. We present a portion of the results we obtained in Figure 15.3 and Table 15.1. Table 15.1 shows an overview of the results we obtained across the different datasets. Our results show clearly that COLIBRI can improve the results of EUCLID significantly on all datasets. On the Restaurant dataset for example, COLIBRI is 6% better than EUCLID on average. On ACM, the average value lies by 4.8%. In the best case, COLIBRI improves the results of EUCLID from 0.85 to 0.99 (Amazon, $ap = 50\%$, KBs = 4). Moreover, COLIBRI never worsens the results of EUCLID. This result is of central importance as it suggests that our approach can be used across the Linked Data Web for any combination of number of knowledge and error rates within the knowledge bases.

The results achieved on the Restaurant dataset are presented in more detail in Figure 15.3. Our results on this dataset (which were corroborated by the results we achieved on the other datasets) show that the results achieved by EUCLID alone depend directly on the probability of errors being introduced into the datasets. For example, EUCLID is able to achieve an F-measure of 0.94 when provided with datasets with an error rate of 30%. Yet, this F-measure sinks to 0.88 when the error rate is set to 50%. These results do suggest that EUCLID is robust against errors. This is due to the approach being able to give properties that contain a small error percentage a higher weight. Still, the COLIBRI results show clearly that COLIBRI can accurately repair the knowledge bases and thus achieve even better F-measures. On this particular data, the approach achieves an F-measure very close to 1 in most cases. Note that the number of iterations required to achieve this score depends directly on the number of knowledge bases and on the error probability.

One interesting observation is that the average F-measure achieved by EUCLID decreases with the number of knowledge bases used for linking. This is simply due to the overall larger number of errors generated by our evaluation framework when the number of knowledge bases is increased. While larger number also make

the detection of errors more tedious, COLIBRI achieves significant increase of F-measure in this setting. In particular, the F-measure of EUCLID is improved upon by up to 12% absolute on the Restaurant dataset ($ap = 50\%$) as well as 7% absolute on Persons1 ($ap = 50\%$).

As expected, the runtime of our approach grows quadratically with the number of knowledge bases. This is simply due to EUCLID being run for each pair of knowledge bases.The runtimes achieved suggest that COLIBRI can be used in practical settings and on large datasets as long as the number of dimensions in EUCLID's search space remains small. In particular, one iteration of the approach on the DBLP datasets required less than 2 minutes per iteration for 3 knowledge bases, which corresponds to 3 EUCLID runs of which each checked 3125 link specifications. The worst runtimes were achieved on the Persons1 dataset, where COLIBRI required up to 11min/iteration. This was due to the large number of properties associated with each resource in the dataset, which forced EUCLID to evaluate more than 78,000 specifications per iteration.

15.5 RELATED WORK

Most LD approaches for learning link specifications developed so far abide by the paradigm of supervised machine learning. One of the first approaches to target this goal was presented in (Isele and Bizer, 2011). While this approach achieves high F-measures, it also requires large amounts of training data. However, creating training data for link discovery is a very expensive process, especially given the size of current knowledge bases. Supervised LD approaches which try to reduce the amount training data required are most commonly based on active learning (see, e.g., (Isele et al., 2012; Ngonga Ngomo et al., 2013)). Still, these approaches are not guaranteed to require a small amount of training data to converge. In newer works, unsupervised techniques for learning LD specifications were developed (Ngonga Ngomo and Lyko, 2013; Nikolov et al., 2012). The main advantage of unsupervised learning techniques is that they do not require any training data to discover mappings. Moreover, the classifiers they generate can be used as initial classifiers for supervised LD approaches. In general, unsupervised approaches assume some knowledge about the type of links that are to be discovered. For example, unsupervised approaches for ontology alignment such as PARIS (Suchanek et al., 2011) aim to discover exclusively `owl:sameAs` links. To this end, PARIS relies on a probabilistic model and maps instances, properties and ontology elements. Similarly, the approach presented in (Nikolov et al., 2012) assumes that a 1-to-1 mapping is to be discovered. Here, the mappings are discovered by using a genetic programming approach whose fitness function is set to a PFM. The main drawback of this approach is that it is not deterministic. Thus, it provides no guarantee of finding a good specification. This problem was addressed by EUCLID (Ngonga Ngomo and Lyko, 2013).

While ontology-matching approaches that rely on more than 2 ontologies have existed for almost a decade (Doan et al., 2003; Euzenat, 2008; Madhavan and Halevy, 2003), LD approaches that aim to discover between n datasets have only started to emerge in newer literature. For instance, the approach proposed by (Hartung et al., 2013) suggests a composition method for link discovery between n datasets. The approach is based on strategies for combining and filtering mappings between resources to generate links between knowledge bases. The framework in-

troduced by (Jiang et al., 2012) aims to predict links in multi-relational graph. To this end, it models the relations of the knowledge bases using set of description matrices and combines them using an additive model. The *Multi-Core Assignment Algorithm* presented by (Böhm et al., 2012) automated the creation of `owl:sameAs` links across multiple knowledge bases in a globally consistent manner. A drawback of this approach is that it requires a large amount of processing power.

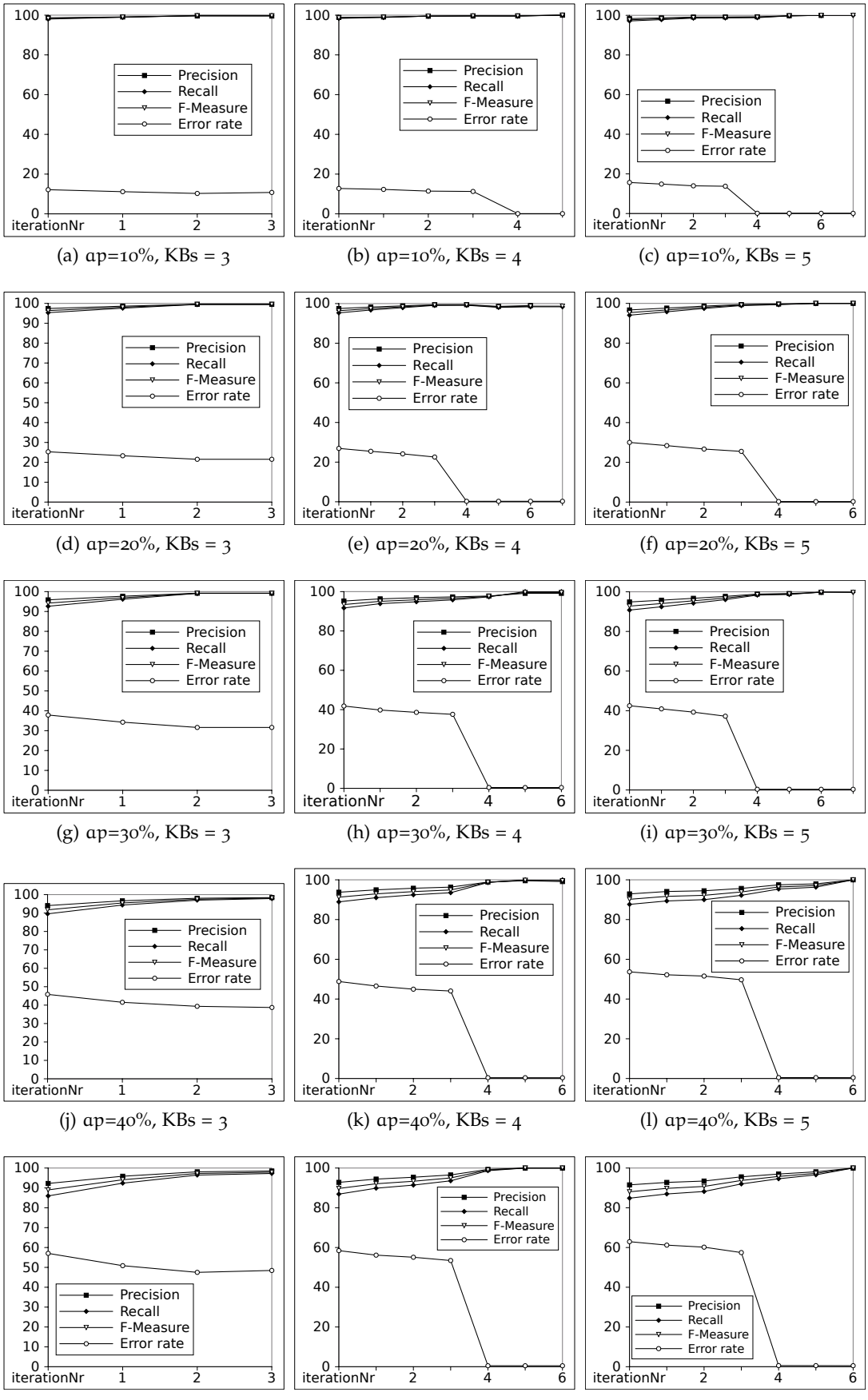
Approaches related to COLIBRI also include link predication approaches based on statistical relational learning (SRL). Examples of SRL approaches that can be used for predicate detection include CP and Tucker (Kolda and Bader, 2009) as well as RESCAL (Nickel et al., 2012), which all rely on tensor factorization. In general, approaches which rely on tensor factorization have a higher complexity that EUCLID. For example, CP's complexity is quadratic in the number of predicates. Related approaches that have been employed on Semantic Web data and ontologies include approaches related to Bayesian networks, inductive learning and kernel learning (Bloehdorn and Sure, 2007; d'Amato et al., 2008; Nickel et al., 2012; Pérez-Solà and Herrera-Joancomartí, 2013; Sutskever et al., 2009). Due to the complexity of the models they rely on, most of these approaches are likely not to scale to very large datasets. LIMES (in which EUCLID is implemented) has yet been shown to scale well on large datasets (Ngonga Ngomo, 2012b). An exact evaluation of the complexity and runtime of a combination of COLIBRI and SRL-based approaches remains future work. More details on SRL can be found in (Getoor and Taskar, 2007).

15.6 CONCLUSION AND FUTURE WORK

We presented COLIBRI, the first unsupervised LD approach which attempts to repair instance knowledge in n knowledge bases ($n \geqslant 2$) to improve its linking accuracy. Our evaluation suggests that our approach is robust and can be used by error rates up to 50% when provided with at least 3 knowledge bases. In addition, our results show that COLIBRI can improve the results of EUCLID by up to 14% F-measure. In future work, we plan to extend our evaluation further and analyse our performance on real data as well as on knowledge bases of different size. We plan to deploy our approach in interactive scenarios within which users are consulted before the knowledge bases are updated. The voting procedure implemented by COLIBRI can be used to provide users with a measure for the degree of confidence in a predicted link and in the need for a repair within an interactive learning scenario.

ap	10%				30%				50%			
Measures	F_E	F_C	R	L	F_E	F_C	R	L	F_E	F_C	R	L
KBs	Restaurant											
3	0.98	1.00	0.6	4	0.94	0.99	0.5	17	0.89	0.98	0.4	43
4	0.99	1.00	1.2	8	0.93	1.00	1.0	33	0.90	1.00	0.9	35
5	0.98	1.00	1.8	20	0.93	1.00	1.5	30	0.88	1.00	1.3	34
KBs	Persons1											
3	0.99	1.00	225.6	11	0.96	1.00	206.2	38	0.94	1.00	190.4	57
4	0.98	1.00	494.3	23	0.96	1.00	422.1	47	0.93	1.00	349.9	77
5	0.98	1.00	819.4	20	0.95	1.00	747.6	75	0.93	1.00	656.2	110
KBs	ACM											
3	0.95	0.96	85.7	220	0.89	0.96	69.3	301	0.84	0.95	66.5	484
4	0.94	0.94	168	12	0.88	0.88	140.4	36	0.83	0.96	131.1	261
5	0.94	0.94	271.7	30	0.87	0.94	240.9	821	0.82	0.84	202.8	348
KBs	DBLP											
3	0.94	0.98	135	220	0.85	0.97	117.2	828	0.77	0.82	111	2686
4	0.93	0.98	268.8	312	0.83	0.90	234.7	306	0.76	0.81	201.1	350
5	0.93	0.98	334.9	517	0.82	0.84	395.9	182	0.76	0.77	338.1	156
KBs	Amazon											
3	0.97	0.99	90.4	60	0.92	0.99	85.2	177	0.86	0.98	81.8	300
4	0.97	0.99	187.5	98	0.91	0.98	172.6	185	0.85	0.99	160.4	150
5	0.96	0.99	301.8	131	0.90	0.99	278.7	369	0.84	0.88	246.8	60

Table 15.1: Average F-measure of EUCLID (F_E) and COLIBRI (F_C) after 10 iterations, runtime (R, in seconds) and number of repaired links L achieved across all experiments. KBs stands for the number of knowledge bases used in our experiments.



(a) ap=10%, KBs = 3 (b) ap=10%, KBs = 4 (c) ap=10%, KBs = 5

(d) ap=20%, KBs = 3 (e) ap=20%, KBs = 4 (f) ap=20%, KBs = 5

(g) ap=30%, KBs = 3 (h) ap=30%, KBs = 4 (i) ap=30%, KBs = 5

(j) ap=40%, KBs = 3 (k) ap=40%, KBs = 4 (l) ap=40%, KBs = 5

(m) ap=50%, KBs = 3 (n) ap=50%, KBs = 4 (o) ap=50%, KBs = 5

Figure 15.3: Overview of the results on the Restaurants dataset

16 A COMPARISON OF SUPERVISED LEARNING APPROACHES

PREAMBLE

In this chapter, we investigate which of the commonly used supervised machine-learning classifiers performs best on the link discovery task. The chapter is taken from (Soru and Ngonga Ngomo, 2014) and presents a work that was supervised by the author, who also co-wrote the paper.

16.1 INTRODUCTION

Over the last years, the development of tools to support the implementation of all Linked Data principles has been regarded as central for the uptake of Linked Data (Auer et al., 2013a). In particular, several tools have been developed to support the fourth Linked Data principle, i.e., the creation of links between datasets. Formally, the general aim of link discovery is to find the set of resource pairs $(s, t) \in S \times T$ such that $R(s, t)$ holds, where R is a given relation such as `owl:sameAs` or `dbp:near`[1] and S and T are sets of resources. Finding this set is a non-trivial task which is commonly approximated by finding the set $M \subseteq S \times T$ of resource pairs such that $\forall (s, t) \in M : sim(s, t) \geqslant \theta$, where $\theta \in [0, 1]$ is a similarity threshold and sim is a complex similarity function; sim consists of a combination of atomic similarity functions σ_i, each of which compares the property values p_i of s (denoted $s.p_i$) and q_i of t (denoted $t.q_i$).

Overall, two main categories of learning algorithms have been implemented so far to support the computation of the set M: *Supervised algorithms* for linking take two sets of resources (called the *source set* and the *target set*) as well as positive and negative examples for links as input. These algorithms then return a set of pairs of resources that they assume to be links. *Unsupervised algorithms* on the other hand only take two sets of resources (i.e., need no training data) and return links. To this end, most of the unsupervised approaches (Ngonga Ngomo and Lyko, 2013; Nikolov et al., 2012) implement an objective function that they aim to optimize. While unsupervised approaches have been shown to perform well on datasets where 1-1 mappings exist between the source and the target sets, most of the current tools implement supervised approaches for link discovery. Still, the performance of the different supervised machine-learning paradigms for link discovery has not been systematically analyzed so far.

We address this research gap by comparing ten different supervised machine-learning paradigms on six different datasets. The goal of this evaluation is to determine the answer to the following questions:

Q_1: Which of the paradigms achieves the average best F-measure?

Q_2: Which of the paradigms is most robust against noise?

Q_3: Which of the methods is the most time-efficient?

1 We use the prefixes defined at `http://prefix.cc`

To this end, we use the well-known n-fold cross-validation experimental setting to evaluate commonly used machine-learning paradigms and present the precision, recall and F-measure they achieve. Moreover, we evaluate their time-efficiency by measuring their runtime over all six datasets. Surprisingly, our results suggest that multilayer perceptrons perform best on the link discovery tasks at hand. Although they have been already used for performing inductive semantic classification (Fanizzi et al., 2009), to the best of our knowledge they have not been tested before on link discovery. From our results, multilayer perceptrons perform best on average of clean data. Logistic regression on the other hand performs best on noisy data. The two approaches are followed by decision-table-based algorithms.

16.2 EVALUATION PIPELINE

We now show each step of the evaluation pipeline. Starting from a source and a target dataset containing individuals, we aim at comparing the performance of different classifiers on the link discovery task. First, the join set between the two dataset is created. Thereafter, the alignment among properties and the perfect mapping among the datasets are loaded to label each instance in the join set and to compute its similarity. Subsequently, a 10-fold cross-validation is applied. Ten supervised machine-learning classifiers are trained on the training sets. Finally, these classifiers are evaluated on the test sets and their scores averaged.

A join between two datasets is basically composed by their Cartesian product, i.e. by all pairs $(s,t) \in S \times T$, where $s \in S$ and $t \in T$. As previously mentioned, each pair in the join represents the input for the similarity function sim. After the join creation, which is a preprocessing phase, property alignments and the perfect mapping are imported. Being $P = \{p_i, \dots p_n\}$ a set of source properties and $Q = \{q_i, \dots q_m\}$ a set of target properties, a property alignment set is an ordered set $A \subseteq P \times Q$. In other words, a source property can be aligned to an arbitrary number of target properties, and vice-versa. For instance, given the property alignments $\{(p_1, q_1), (p_1, q_2)\}$, the overall atomic similarity value is the mean value of atomic similarities $\sigma(s.p_1, t.q_1)$ and $\sigma(s.p_1, t.q_2)$. Property alignments are usually carried out by domain experts, as well as the perfect mapping. For each example (s,t), we evaluate the atomic similarities among datatype values $(s.p, t.q)$ for all $(p,q) \in A$. In case properties (p,q) are object properties, the alignment holds among the properties connected to the respective object nodes. String values were compared using weighted trigram and cosine similarity, as well as weighted edit distance. In this paper, we refer to the definition of weighted edit distance proposed in (Soru and Ngonga Ngomo, 2012) using error probabilities on confusion matrices as weights (see (Soru and Ngonga Ngomo, 2013)). Numerical values were compared using a logarithm-based similarity. After being computed, all similarity values are normalized into the interval $[0,1]$.

16.3 EVALUATION

16.3.1 *Experimental Setup*

We evaluated ten machine learning techniques against six different datasets. Four classifiers belong to the linear non-probabilistic class (*linear* and *polynomial support vector machines*, with and without *sequential minimal optimization (SMO)* and *linear*

regression), four of them rely on probability theory (*logistic regression*, *naïve Bayes*, *random tree* and *J48*), one of them is based on neural networks (*multilayer perceptron*) and one is rule-based (*decision table*). The first three datasets `D1`, `D2` and `D3` belong to the Ontology Alignment Evaluation Initiative (OAEI) 2010 Benchmark[2] and are synthetic in nature. In order to verify the scalability of the algorithms as well as their performance on real data, we evaluated them on three larger datasets which contained real data. Dataset `D4`, which contains over 6 million joins, belongs to the bibliographic domain, while `D5` and `D6` belong to the e-commerce domain. The datasets `D4`, `D5` and `D6` are part of the Benchmark for Entity Resolution[3] (Köpcke et al., 2010). The link discovery community relies on the information retrieval concepts of precision, recall and F_1-measure. We will thus report the *F_1-measure* (or *F-score*) in the following.

The Java source code used within this evaluation is available online[4]. Among others, we utilized LibSVM[5] for the linear and polynomial support vector machines and the Weka Java APIs[6] for the remaining eight machine learning techniques. For SVMs, we found the following values using a grid search: $C = 10^6$ for `D1:D4`, $C = 10^3$ for `D5:D6`, $\epsilon = 10^{-3}$, $\gamma = 1$. Moreover, we chose a third-degree polynomial kernel ($d = 3$). We used a 10-fold cross-validation across all experiments and set the test significance level to .05. All experiments were carried out on a 64-bit Ubuntu Linux machine with 8GB of RAM and a 2.5 GHz CPU.

16.3.2 *Experimental Results*

The F-measures achieved by all approaches at hand are shown in Table 16.1.[7] As seen, all classifiers performed excellently on dataset `D1`, achieving high F-scores above 97%. All classifiers except naïve Bayes (~35%) performed well on dataset `D2`. Noteworthy is the perfect classification (100%) carried out by Linear SMO on `D1` and the decision table algorithm on `D2`. The same excellent result has been achieved on dataset `D3` by multilayer perceptrons, linear regression and decision tables. Here, only the random tree classifier remains below 90% F-score. Dataset `D4` reflects the results obtained on `D2`: While the naïve Bayes classifier plays as lonely outlier (~29%), all other classifiers reach an F-score between 92% and 98%. Results on both e-commerce datasets (`D5` and `D6`), allow to analyse the behaviour of the algorithms considered herein on noisy datasets that are very hard to learn (Köpcke et al., 2010; Soru and Ngonga Ngomo, 2012). The fifth dataset highlights a drawback on the classification task for naïve Bayes classifiers (~3%). Interestingly, while basic linear SVMs perform poorly (~27%), SVMs with SMO outperform the other classifiers. Being SVM and SMO two approaches to the same problem, we consider the different implementation from the respective libraries as the reason of this result. The last dataset follows the average trend, except for the decision table classifier, which ranks second-last with less than 30% F-score. Multilayer perceptron sets the highest performance result with ~43%, tailed by logistic regression (~42%) and random tree (~41%). The difficulty to linearly separate few and sparse

2 http://oaei.ontologymatching.org/2010/benchmarks
3 http://goo.gl/bvWBjA
4 https://github.com/mommi84/BALLAD
5 http://www.csie.ntu.edu.tw/~cjlin/libsvm
6 http://www.cs.waikato.ac.nz/ml/weka
7 Further results were collected and inserted into a technical report at http://mommi84.github.io/BALLAD

positives among a much higher number of negatives led many classifiers to fail on D5-D6.

Rank	Classifier	D1	D2	D3	D4	D5	D6	Average
1.	Multilayer Perceptron	99.50%	99.50%	**100.00%**	97.43%	35.58%	**43.49%**	**79.25%**
2.	Logistic Regression	99.90%	98.12%	96.67%	97.71%	40.64%	41.92%	79.16%
3.	Linear SMO	**100.00%**	98.73%	**100.00%**	92.58%	**46.63%**	31.39%	78.22%
4.	Decision Table	97.98%	**100.00%**	**100.00%**	97.66%	42.44%	29.66%	77.96%
5.	J48	99.50%	95.56%	98.29%	97.66%	44.28%	31.53%	77.80%
6.	Linear Regression	99.30%	96.92%	**100.00%**	96.36%	37.06%	36.84%	77.75%
7.	Random Tree	97.45%	99.24%	89.89%	96.82%	39.38%	41.03%	77.30%
8.	Linear SVM	99.40%	98.99%	97.75%	**97.81%**	27.06%	39.18%	76.70%
9.	Polynomial-3 SVM	99.40%	93.76%	98.29%	97.67%	37.28%	31.69%	76.35%
10.	Naïve Bayes	97.75%	35.05%	95.19%	29.47%	2.92%	11.90%	45.38%

Table 16.1: F_1-scores for ten different classifiers on six test datasets are shown, ranked by average F_1-score

Linear and polynomial SVM appeared to be one order of magnitude faster than the other approaches on D1-D2. The same performance was achieved by random trees and linear SVM on D4 and by J48 on D5, while polynomial SVM was found to be significantly slower than the average trend on D5-D6. The computation runtime for the fastest classifier obtained an order of magnitude of 1*s* (D3), 10*s* (D1-D2-D6) and 100*s* (D4-D5) respectively. A detailed runtime table is shown in the technical report stated above.

Table 16.1 is sorted by average F-score, which can be seen in the last column. Multilayer perceptrons performed best among all, reaching the top place both including (79.25%) and excluding (99.11%) the noisier datasets D5 and D6. Again excluding these datasets, decision tables ranked second (98.91%). However, logistic regression outperformed decision tables on overall average (79.16% against 77.96%), as well as linear SMO (78.22%). Linear regression obtained almost the same results as linear SMO, decision tables and random trees, although random trees achieved 3% less if we exclude noisy datasets. On average, linear SVM (76.70%) carried out a better classification than polynomial SVM (76.35%). Naïve Bayes had the worst results in both cases, reaching only an average F-score of 45.38%. The dependence among property values is likely one of the reasons of such failure, especially on real datasets. In fact, a Naïve Bayes classifier considers all features as independent from each other.

Overall, our results hint that while some average trends for machine-learning-based link discovery can be suggested, no algorithm outperforms all other significantly. The results on the non-noisy datasets suggest that multilayer perceptrons should be used as default learning approach for most datasets. Should they fail, we suggest using linear SVM and then decision tables. This answers Q_1. The results on the noisy datasets suggest that random trees and multilayer perceptrons are most robust against noise and answer Q_2. Thus, we suggest using these two approaches first when confronted with noise before contemplating other approaches. Finally, while all approaches scale well and random trees are fastest on 3 of the datasets we considered, we still suggest using multilayer perceptrons as they achieve acceptable runtimes overall. This answers Q_3.

16.3.3 *Comparison with existing frameworks*

We compared our results with state-of-the-art frameworks which have been run on the six test datasets. The first three evaluation results belong to the OAEI2010 initiative, wherein the instance-matching implementation of the refalign (Schadd and Roos, 2011) framework won the *benchmark* contest carrying out a perfect matching on the synthetic datasets `D1:D3`. ASMOV (Jean-Mary et al., 2010) and AgreementMaker (Cruz et al., 2010) achieved excellent results (100%; 99%) on `D1` and good results (94%; 89%) on `D2`, however the latter had a poor accuracy on the third dataset (70%). The unsupervised version of the EAGLE approach (Ngonga Ngomo and Lyko, 2012) yields state-of-the-art results on the synthetic datasets and performs best on `D4`. MARLIN (Bilenko and Mooney, 2003) yielded better results using SVM than using AD-Tree on all real datasets. Although their scores are satisfiable, none of these approaches achieved state-of-the-art accuracies. ACIDS (Soru and Ngonga Ngomo, 2012), COSY (an undisclosed commercial system for entity resolution), and FEBRL SVM (Christen, 2008) currently represent the state of the art for `D4`, `D5`, and `D6` respectively. Our results showed that even using simple settings, some classifiers presented in this paper achieved state-of-the-art results on `D1:D4`. Moreover, while single machine-learning approaches perform well on single tasks, the combination of several classifiers, kernels and techniques can achieve optimized results and better accuracies.

16.4 RELATED WORK

Overall, two main problems need to be tackled to address the tool support of link discovery. First, naïve implementations of link discovery approaches have a quadratic *time complexity*. This problem has been addressed in manifold ways: An extensive amount of literature has been published by the database community (see (Elmagarmid et al., 2007; Köpcke et al., 2010) for surveys). In particular, time-efficient deduplication algorithms such as PPJoin+ (Xiao et al., 2008), EDJoin (Xiao et al., 2008), PassJoin (Li et al., 2011) and TrieJoin (Wang et al., 2010) were developed over the last years. Several of these were then integrated into the hybrid link discovery framework LIMES (Ngonga Ngomo, 2012b). Moreover, dedicated time-efficient approaches were developed for link discovery. For example, RDF-AI (Scharffe et al., 2009) implements a five-step approach that comprises the preprocessing, matching, fusion, interlink and post-processing of datasets. (Ngonga Ngomo and Auer, 2011) presents an approach based on the Cauchy-Schwarz inequality that allows discarding a large number of unnecessary computations. The approaches HYPPO (Ngonga Ngomo, 2011) and $\mathcal{HR}^3$ (Ngonga Ngomo, 2012a) rely on space tiling in spaces with measures that can be split into independent measures across the dimensions of the problem at hand. Standard blocking approaches were implemented in the first versions of SILK and later replaced with MultiBlock (Isele et al., 2011), a lossless multi-dimensional blocking technique. KnoFuss (Nikolov et al., 2012) also implements blocking techniques to achieve acceptable runtimes.

In addition to addressing the runtime of link discovery, several machine-learning approaches have been developed to learn link specifications (also called linkage rules) for link discovery. For example, machine-learning frameworks such as FEBRL (Christen, 2008) and MARLIN (Bilenko and Mooney, 2003) rely on models such as Support Vector Machines (Cristianini and Ricci, 2008) and decision trees (Safavian

and Landgrebe, 1991) to detect classifiers for record linkage. RAVEN (Ngonga Ngomo et al., 2011) combines active learning version and perceptron learning to detect linear or Boolean classifiers. The EAGLE approach (Ngonga Ngomo and Lyko, 2012) combines active learning and genetic programming to detect link specifications. KnoFuss (Nikolov et al., 2012) goes a step further and presents an unsupervised approach based on genetic programming for finding accurate link specifications.

16.5 CONCLUSION

In this paper, we presented a systematic evaluation of ten different classifiers for supervised learning on the link discovery task. We use three real and three artificial datasets to compare the approaches. On average, all approaches apart from Naïve Bayes perform well. Still, the results show that multilayer perceptrons are best on average. This result is interesting because to the best of our knowledge, multilayer perceptrons have not been used for link discovery so far. We will thus integrate them into the LIMES framework and make them available via the SAIM interface. Moreover, the performance on linear regression on real datasets make it a viable alternative to multilayer perceptrons when dealing with real-life problems. Interestingly, the behavior of the different approaches on the datasets used in our evaluation suggests that they are complementary. Thus, in order to achieve even better accuracy, we aim to combine them by using ensemble learning techniques. We will investigate how good the same techniques perform in case of small training sets, eventually utilizing a larger amount of similarity measures. Finally, an avenue of interest would be to incorporate a component based on Statistical Relational Learning to exploit information about the RDF graph structure while linking.

17 EFFICIENT KEY DETECTION FOR LINK DISCOVERY

PREAMBLE

In contrast to structured data published in relational databases (where a key is often provided explicitly), finding a set of properties that allows identifying a resource uniquely is a non-trivial task. Still, keys can be used when improving the runtime of link discovery frameworks. This chapter presents a refinement operator for key discovery. The corresponding paper was presented in (Soru et al., 2015b). The author designed the refinement operator and proved its characteristics in collaboration with the first author. Moreover, the author co-designed the evaluation, co-wrote the paper and supervised the rest of the work.

17.1 INTRODUCTION

The number of facts published in the Linked Data Web has grown considerably over the last years (Auer et al., 2013a). In particular, large knowledge bases such as LinkedTCGA (Saleem et al., 2013) and LinkedGeoData (Stadler et al., 2012b) encompass more than 20 billion triples each. The architectural principles behind the Linked Data Web are akin to those on the Web. In particular, the decentral data publication process leads to facts on the same real-world entities being published across manifold knowledge bases. For example, information on *Austin, Texas* is distributed across several knowledge bases, including DBpedia[1], LinkedGeoData and GeoNames.[2] Given the size of the current linked datasets, providing unique means to characterize resources within existing datasets would facilitate the use of these knowledge bases, for example within the context of entity search, data integration, Linked Data compression and link discovery (Pernelle et al., 2013). Especially for the link discovery task, being provided with unique descriptions of resources in a knowledge base would allow for the more time-efficient computation of property matchings for link specifications, a task that has been pointed out to be particularly tedious in previous work (Cheatham and Hitzler, 2013).

In relational databases, keys are commonly either artificial or sets of columns that allow to describe each resource uniquely. Previous works (Atencia et al., 2014; Pernelle et al., 2013; Symeonidou et al., 2014) adopt this approach for uniquely describing RDF data and use properties instead of columns. Several problems occur when trying to detect keys for RDF data.

1. Resources from the same datasets might not all have the same properties. For example, in the fragment of DBpedia 3.9 shown in Figure 17.1, only 50% of the resources have a `:meshNumber`. Thus, while the `:meshNumber` is unique, it cannot be used as a key for this dataset.

2. The inverse problem exists for the `:graySubject`, which covers all resources but is not unique as the trigeminal nerve and the lacrimal nerve have the

1 http://dbpedia.org
2 http://www.geonames.org/

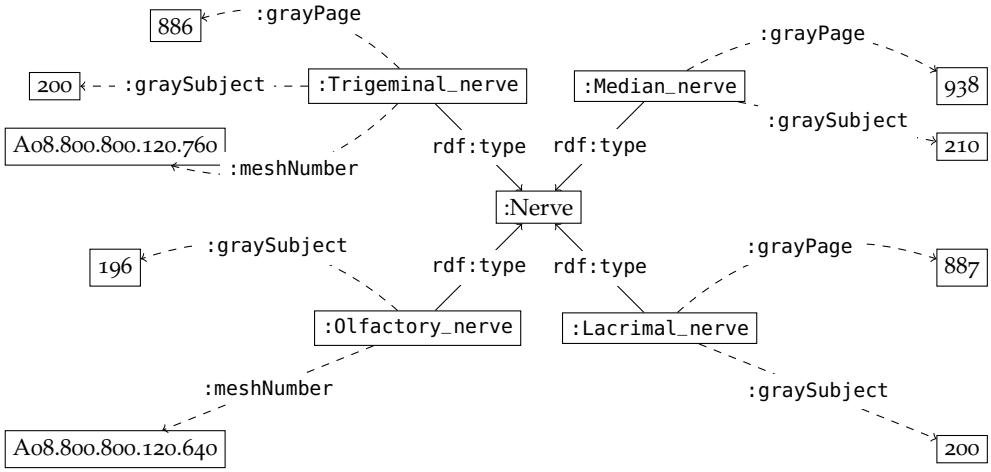


Figure 17.1: Fragment from a knowledge base on human nerves. The fragment was extracted from DBpedia 3.9.

same `:graySubject`. For our toy dataset, only keys of size larger that 1 exist (e.g., {`:graySubject`, `:grayPage`}).

3. The key discovery problem is exponential in the number of properties n in the knowledge base, as the solution space contains $2^n - 1$ possible sets of keys. Thus, naïve solutions to the key discovery problem do not scale.

Moreover, depending on the use case, key discovery approaches have to be able to detect a single key (e.g., to link resources within a knowledge base) or to detect all keys for a resource (e.g., when integrating data across knowledge bases).

In this chapter, we address the three problems of key discovery within both settings of key discovery (i.e., finding all keys or almost-keys within a given threshold) by using a refinement operator dubbed ρ. This operator is able to detect sets of properties that describe any instance of a given class in a unique manner. By these means, it can generate n-tuples of property values that can be used as keys for resources which instantiate a given class. Our operator relies on a scoring function to compare sets of properties. Based on this comparison, it can efficiently detect single keys, all keys and even predict whether a key can be found in a given dataset. In addition to being finite, non-redundant and proper, our operator also scales well and can thus be used on large knowledge bases. Our contributions are:

- We provide the first refinement operator for key discovery on RDF knowledge bases.
- We prove that our operator is finite, non-redundant, proper, but not complete.
- We utilize the combination of a hash index to compute the discriminability score, i.e. the ability for a set of properties to distinguish their subjects, with two monotonicities of keys to prune the refinement tree and thus ensure that our operator scales.
- We show that our approach succeeds on datasets where current state-of-the-art approaches fail.

- We evaluate our operator on the OAEI instance matching benchmark datasets as well as on DBpedia classes with large populations. In particular, we measure the overall runtime, the memory consumption and the reduction ratio of our approach. Our results suggest that we outperform the state of the art w.r.t. correctness and memory consumption. Moreover, our results suggest that our approach terminates within an acceptable time frame even on very large datasets.

The rest of this chapter is structured as follows: We begin by defining the problem at hand formally. Thereafter, we present our operator and prove its theoretical characteristics. After a discussion of related work, we evaluate our operator on synthetic and real data. We then conclude and present some future work.

17.2 PRELIMINARIES

In the following, we formalize the definition of keys that underlie this chapter. This definition is used by our refinement operator to efficiently detect keys.

17.2.1 *Keys*

Let K be a finite RDF knowledge base containing instances which belong to a given class and their Concise Bounded Description (CBD).[3] K can be regarded as a set of triples $(s, p, o) \in (\mathcal{R} \cup \mathcal{B}) \times \mathcal{P} \times (\mathcal{R} \cup \mathcal{L} \cup \mathcal{B})$, where $\mathcal{R}$ is the set of all resources, $\mathcal{B}$ is the set of all blank nodes, $\mathcal{P}$ the set of all predicates and $\mathcal{L}$ the set of all literals. We call two resources $r_1, r_2 \in \mathcal{R}$ distinguishable w.r.t. a set of properties $P = \{p_1, \dots p_n\}$ iff $\exists p \in \{p_1, \dots p_n\} : \neg((r_1, p, o) \wedge (r_2, p, o))$. Given a knowledge base K, the idea behind *key discovery* is to find one or all sets of properties which make their respective subjects distinguishable in K.

Definition 10 (Key)**.** *We call a set of properties* $P \subseteq \mathcal{P}$ *a* key *for a knowledge base* K *(short: key, denoted* $\mathrm{key}(P, K)$*) if all resources in* K *are distinguishable w.r.t.* P*.*

Definition 11 (Minimal key)**.** *We call* P *a* minimal key *(short: mkey) iff* P *is a key but none of its subsets is. Formally,*

$$\mathrm{mkey}(P, K) \Rightarrow \mathrm{key}(P, K) \wedge (\neg \exists P' \subset P : \mathrm{key}(P', K)). \tag{17.1}$$

17.2.2 *Discriminability*

A *key* for an RDF knowledge base and a *primary composite key* for a database share the same aim. RDF properties represent the projection of database fields into the RDF paradigm, as well as each resource represents a tuple. However, while a tuple element has only one single value, a property might link a resource to more than one RDF objects. Therefore, two resources are distinguishable from each other w.r.t. a set of properties P if their sets of objects are different for at least one $p \in P$.

To the best of our knowledge, this particular feature of keys was not taken into account by previous works on key discovery for RDF data (Atencia et al., 2014; Pernelle et al., 2013; Symeonidou et al., 2014). For instance, (Pernelle et al., 2013; Symeonidou et al., 2014) consider two resources r and r' as not distinguishable w.r.t. P if for each $p \in P$ they share at least one object.

3 For the definition of CBD, see http://www.w3.org/Submission/CBD/.

Dataset D1:
d1:Film(f1), $d1{:}hasActor(f1,''B.Pitt'')$, $d1{:}hasActor(f1,''J.Roberts'')$,
$d1{:}director(f1,''S.Soderbergh'')$, $d1{:}releaseDate(f1,''3/4/01'')$, $d1{:}name(f1,''Ocean's\ 11'')$,
d1:Film(f2), $d1{:}hasActor(f2,''G.Clooney'')$, $d1{:}hasActor(f2,''B.Pitt'')$,
$d1{:}hasActor(f2,''J.Roberts'')$, $d1{:}director(f2,''S.Soderbergh'')$, $d1{:}director(f2,''P.Greengrass'')$,
$d1{:}director(f2,''R.Howard'')$, $d1{:}releaseDate(f2,''2/5/04'')$, $d1{:}name(f2,''Ocean's\ 12'')$
d1:Film(f3), $d1{:}hasActor(f3,''G.Clooney'')$, $d1{:}hasActor(f3,''B.Pitt'')$
$d1{:}director(f3,''S.Soderbergh'')$, $d1{:}director(f3,''P.Greengrass'')$, $d1{:}director(f3,''R.Howard'')$,
$d1{:}releaseDate(f3,''30/6/07'')$, $d1{:}name(f3,''Ocean's\ 13'')$,
d1:Film(f4), $d1{:}hasActor(f4,''G.Clooney'')$, $d1{:}hasActor(f4,''N.Krause'')$,
$d1{:}director(f4,''A.Payne'')$, $d1{:}releaseDate(f4,''15/9/11'')$, $d1{:}name(f4,''The\ descendants'')$,
$d1{:}language(f4,''english'')$
d1:Film(f5),$d1{:}hasActor(f5,''F.Potente'')$, $d1{:}director(f5,''P.Greengrass'')$,
$d1{:}releaseDate(f5,''2002'')$, $d1{:}name(f5,''The\ bourne\ Identity'')$, $d1{:}language(f5,''english'')$
d1:Film(f6),$d1{:}director(f6,''R.Howard'')$, $d1{:}releaseDate(f6,''2/5/04'')$,
$d1{:}name(f6,''Ocean's\ twelve'')$

Figure 17.2: Example of RDF data, as reported in (Symeonidou et al., 2014)

Figure 17.2 shows an example of RDF data, as reported in (Symeonidou et al., 2014). Here, the authors claim that $\mathsf{P} = \{p_1\} = \{\texttt{:hasActor}\}$ is not a key because `"G.Clooney"` is the object of more than one instance of `:Film`. We would instead consider P as a key, since every film is linked with a different set of objects (sobj), i.e.:

$\text{sobj}(\texttt{:f1}, p_1) = \{\texttt{"B.Pitt"}, \texttt{"J.Roberts"}\}$
$\text{sobj}(\texttt{:f2}, p_1) = \{\texttt{"G.Clooney"}, \texttt{"B.Pitt"}, \texttt{"J.Roberts"}\}$
$\text{sobj}(\texttt{:f3}, p_1) = \{\texttt{"B.Pitt"}, \texttt{"G.Clooney"}\}$
$\text{sobj}(\texttt{:f4}, p_1) = \{\texttt{"G.Clooney"}, \texttt{"N.Krause"}\}$
$\text{sobj}(\texttt{:f5}, p_1) = \{\texttt{"F.Potente"}\}$
$\text{sobj}(\texttt{:f6}, p_1) = \varnothing$

Note that `:f6` is still distinguishable from the other resources w.r.t. P, since no other instance of `:Film` in the knowledge base has 0 actors. This particular case was not considered, for example, by the authors of (Atencia et al., 2014).

17.2.3 *Properties of a key*

Keys abide by several monotonicities Pernelle et al. (2013). The first is the so-called *key monotonicity*, which is given by

$$\text{key}(\mathsf{P}, \mathsf{K}) \Rightarrow \forall \mathsf{P}' : \mathsf{P} \subseteq \mathsf{P}' \Rightarrow \text{key}(\mathsf{P}', \mathsf{K}). \tag{17.2}$$

The reciprocal monotonicity is called the *non-key monotonicity*, which is given by

$$\neg\text{key}(\mathsf{P}, \mathsf{K}) \Rightarrow (\forall \mathsf{P}' \subseteq \mathsf{P} : \neg\text{key}(\mathsf{P}', \mathsf{K})). \tag{17.3}$$

In other words, adding a property to a key yields another key, whilst removing a property to a non-key yields another non-key. In this chapter, we present a key discovery approach based on refinement operators.

17.3 A REFINEMENT OPERATOR FOR KEY DISCOVERY

In this section, we present our refinement operator for key discovery and prove some of its theoretical characteristics. Our formalization is based on that presented in (Lehmann and Hitzler, 2010).

Let $P \subseteq \mathcal{P}$. Moreover, let $score : 2^{\mathcal{P}} \to [0, 1]$ be a function that maps each subset P of $\mathcal{P}$ to the fraction of subject resources from K that are distinguishable by using P.

Theorem 1 (Induced quasi-ordering). *The score function induces a quasi-ordering $\preceq$ over the set $\mathcal{P}$, which we define as follows:*

$$P_1 \preceq P_2 \Leftrightarrow \min_{p \in P_1} score(p) \leqslant \min_{q \in P_2} score(q). \tag{17.4}$$

The reflexivity and transitivity of $\preceq$ are direct consequences of the reflexivity and transitivity of $\leqslant$ in $\mathbb{R}$. Note that $\preceq$ is not antisymmetric as two sets of properties P_1 and P_2 can be different and contain the property with the lowest score, leading to $P_1 \preceq P_2$ and $P_2 \preceq P_1$.

Definition 12 (Refinement Operator). *Given a quasi-ordered space (S, op) an upward refinement operator r is a mapping from S to 2^S such that $\forall s \in S : s' \in r(s) \Rightarrow op(s, s')$. s' is then called a generalization of s.*

We define our refinement operator over the space $(\mathcal{P}, \preceq)$. First, we begin by ordering the elements of $\mathcal{P}$ according to their score in ascending order, i.e., $\forall p_i, p_j \in \mathcal{P}, i \leqslant j \Rightarrow score(p_i) \leqslant score(p_j)$. Then, we can define our operator as follows:

$$\rho(P) = \begin{cases} \mathcal{P} \text{ iff } P = \emptyset, \\ \{P \cup \{p_1\}, \ldots, P \cup \{p_i\}\} \text{ where } p_j \in P \Rightarrow i < j. \end{cases} \tag{17.5}$$

For example, the complete refinement graph for $\mathcal{P} = \{p_1, p_2, p_3\}$ is given in Figure 17.3. We use this operator in an iterative manner by only expanding the node with the highest score in the refinement graph. The intuition behind this approach to searching for key is that by ordering properties by their score, we can easily detect and expand the most promising sets of properties without generating redundant nodes. To prove some of the characteristics of ρ, we need to explicate the concept of a refinement chain:

Definition 13 (Refinement chain). *A set $P_2 \in \mathcal{P}$ belong to the refinement chain of $P_1 \in \mathcal{P}$ iff $\exists k \in \mathbb{N} : P_2 \in \rho^k(P_1)$, where $\rho^k(P) = \begin{cases} \mathcal{P} \text{ iff } k = 0, \\ \rho(\rho^{k-1}(P)) \text{ else .} \end{cases}$*

For example, a refinement chain exists between $\{p_3\}$ and $\{p_1, p_2, p_3\}$ in the example shown in Figure 17.3. There is yet no refinement chain between $\{p_1\}$ and $\{p_2\}$ in the same example.

A refinement operator r over the quasi-ordered space (S, op) can abide by the following criteria.

Definition 14 (Finiteness). *r is finite iff $r(s)$ is finite for all $s \in S$.*

Definition 15 (Properness). *r is proper if $\forall s \in S, s' \in r(s) \Rightarrow s \neq s'$.*

Definition 16 (Completeness). *r is said to be complete if for all s and s', $op(s', s)$ implies that there is a refinement chain between s and s'.*

Definition 17 (Redundancy). *A refinement operator r over the space (S, op) is redundant if two different refinement chains can exist between $s \in S$ and $s' \in S$.*

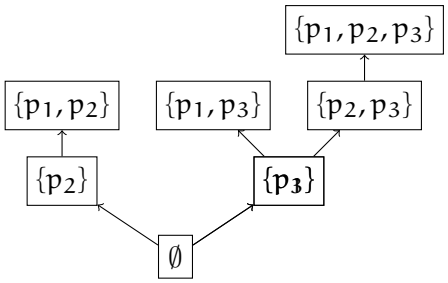


Figure 17.3: Complete refinement graph for $\mathcal{P} = \{p_1, p_2, p_3\}$. The nodes of the graph are subsets of $\mathcal{P}$. A directed edge (a, b) means $b \in \rho(a)$.

In the following, we show that ρ is finite, proper and non-redundant but not complete.

Theorem 2. *ρ is finite when applied to a finite knowledge base K.*

Proof. The finiteness of ρ is a direct result of K being finite. The upper bound of the number of properties in K is the number of triples in K. Thus, $|K| < \infty \Rightarrow |\mathcal{P}| < \infty$. Per definition, $|\rho(P)| \leqslant |\mathcal{P}|$. Thus, we can conclude that $\forall P \in \mathcal{P} : |\rho(P)| < \infty$. □

Theorem 3. *ρ is proper.*

Proof. The properness of ρ also results from the definition of ρ. As we always add exactly a property to P when computing $\rho(P)$, we know that $|\rho(P)| = |P| + 1$. Thus, $\rho(P) \neq P$ must hold. □

Theorem 4. *ρ is not complete.*

Proof. The incompleteness of ρ follows from the definition of $\rho(\emptyset)$. Let $\mathcal{P} = \{p_1, \ldots, p_n\}$. Then $\{p_1\} \preceq \{p_n\}$. Yet, there is clearly no refinement chain between $\{p_n\}$ and $\{p_1\}$ as any subset of $\mathcal{P}$ connected to p_n via a refinement chain must have a magnitude larger than one. Yet, the magnitude of $\{p_1\}$ is 1, which shows that $\{p_1\}$ cannot be connected to $\{p_n\}$ via a refinement chain. □

Theorem 5. *ρ is not redundant.*

Proof. ρ being redundant would mean that a pair of property sets (P, P') exist, where P is linked to P' by two distinct refinement chains C_1 and C_2. Given that these two chains begin and end at the same node, there must be a node N that is common to the two chains but has two distinct fathers N_1 and N_2 that are such that N_1 belongs to C_1 and not to C_2 while N_2 belong to C_2 but not to C_1. Now, N_1 being the father of N means $\exists p_i \in \mathcal{P} : N = N_1 \cup p_i$. Conversely, N_2 being the father of N also means that $\exists p_j \in \mathcal{P} : N = N_2 \cup p_j$. Now if $N_1 \neq N_2$, then we can assume wlog that $i < j$. For $N \in \rho(N_2)$ to hold, j resp. i must be less than the index of any element of N_2 resp. N_1. Moreover, for $N_1 \cup p_i = N_2 \cup p_j$ to hold, p_i would have to already be a element of N_2. However, by virtue of the construction of ρ, this means that $(N_2 \cup \{p_j\}) \notin \rho(N_2)$ given that $p_i \in N_2$ and $i < j$. Thus, we can conclude that there cannot be any to distinct refinement pairs between two subsets of $\mathcal{P}$. □

17.4 APPROACH

In this section, we present **ROCKER**, a refinement operator approach for key discovery. Our approach was designed with scalability in mind. To this end, it implements a scalable version of the discriminability score function based on a hash index. Moreover, we use the monotonicities of keys to check for the existence of keys as well as decide on nodes that need not be refined.

17.4.1 *Implementation*

In order to increase the scalability of our operator, we chose an hybrid approach using both in-memory and disk storage database. The following tasks are then performed by ROCKER:

1. the knowledge base model is built using the Apache Jena library;
2. for each class, its instances and properties are extracted;
3. this information is stored and indexed over a *B-tree* structure, whereas the instance URI is used as a key;
4. object values are sorted alphabetically, imploded into a string, indexed using hash codes, and stored into each tuple element;
5. the refinement operator starts from the empty-set node;
6. at each node, the discriminability score is computed by performing a query to the database;
7. the computation terminates according to some rule.

17.4.2 *Definition of the score function*

We firstly define the set of subjects of the knowledge base, i.e. the instances of a given class.

$$S = \{s : \exists (s, p, o) \in K\} \tag{17.6}$$

We then provide the definition of discriminability for two resources w.r.t. a set of properties P.

$$\mathrm{discr}(s, s', P) \Leftrightarrow \forall s \in S \; \forall p \in P \; \nexists s' \in S : \mathrm{sobj}(s, p) \equiv \mathrm{sobj}(s', p) \tag{17.7}$$

where sobj is the "set of objects" function introduced in Section 17.2.2. Finally, we define the score of P (denoted $\mathrm{score}(P)$) as the number of subject resources of K that are distinguishable w.r.t. P by using the following formula:

$$\mathrm{score}(P) = \frac{|\{s \in S : \forall s' \in S \; s \neq s' \Rightarrow \mathrm{discr}(s, s', P)\}|}{|S|}. \tag{17.8}$$

The set P is a key if $\mathrm{score}(P) = 1$, i.e., if P covers all subject resources from K and all resources are distinguishable w.r.t. P.

Basing the refinement on this scoring function has the advantage of allowing ROCKER to cover not only keys but also k-almost-keys (Symeonidou et al., 2014),

which are defined as follows: P is a k-almost-key if $\exists X \subseteq S : |X| \leqslant k \wedge \forall s, s' \in S \backslash X : \mathrm{discr}(s, s', P)$. Consider for example the data shown in Figure 17.2. If `:f2` did not have `"J. Roberts"` in the list of its actors, then it would not be distinguishable from `:f3`. In this case, the set {`hasActor`} would be 2-almost-key. We can derive a minimal score α for a k-almost-key by simply using the maximal magnitude of X within $\mathrm{score}(P)$:

$$|X| \leqslant k \rightarrow \mathrm{score}(P) \geqslant \frac{|S| - k}{|S|} = \alpha. \tag{17.9}$$

Note that a key is a 0-almost-key. Moreover, every k-almost-key with $k \geqslant 0$ is also a $(k+1)$-almost-key.

In our implementation, the score function for P is thus calculated by querying the class table for the columns associated with the properties in P. For each row, the returned values are then concatenated and added to a sorted set, where duplicates are discarded automatically by virtue of the definition of set. The size of this final set represents the numerator for Equation 17.8.

17.4.3 *Refinement Operator*

The pseudo code of ROCKER's refinement operator is shown in Algorithm 17.1. Given a set of triples K and a set of properties $\mathcal{P}$, we begin by checking whether our ρ-based approach is able to find a key at all. This can be done by computing $\mathrm{score}(\mathcal{P})$. If the score is less than 1 (i.e., if $\mathcal{P}$ is not a key), then we know no key exists by virtue of the non-key monotonicity. We can thus terminate and return $\varnothing$, unless a threshold $\alpha < 1$ is given. Under this setting, we terminate if $\mathrm{score}(\mathcal{P}) < \alpha$. Now if $\mathcal{P}$ is a key, then some of its subsets might be minimal keys. We then begin by sorting the elements of $\mathcal{P}$ w.r.t. their score. This heuristic tries to make the refinement operator discover keys earlier, so that their descendants can be pruned from the refinement tree, thus decreasing the number of score calculations. A maximal-priority queue is then initialized, where the priority of an element is its score. The queue is initialized with the empty set with a priority of 0. We then take the element P of the queue with the highest priority iteratively and remove it from the queue. Thank to the non-redundancy of ρ, there is no need to check whether P has been seen before. P is refined to P', whose elements p are then checked iteratively. We thus evaluate the scores of all elements of P'. If their score is less than 1, then they are added to the queue. If their score is 1, then we add them to the solution and do not add them to the queue, as they do not need to be refined any further by virtue of the key monotonicity. We then return the set of all keys.

Our approach has several advantages due to the theoretical characteristics of ρ and the key monotonicities:

1. It terminates quickly if there is no key by virtue of using the non-key monotonicity.
2. It is guaranteed to find all existing minkeys by virtue of the key monotonicity.
3. Using a sorted queue, it encourages node pruning by evaluating the most promising nodes first.
4. It never visits the same node twice due to the non-redundancy of ρ.

5. It is ensured to find all existing keys.

Algorithm 17.1 ROCKER's algorithm for detecting all keys. The algorithm for detecting a single key does not require the solution variable. Instead, it returns the first P having $score(P) = 1$ it finds.

Require: Set of triples K
1: $\mathcal{P} = \{p : \exists s, p \text{ with } (s, p, o) \in K\}$
2: **if** $score(\mathcal{P}) < 1$ **then**
3: **return** $\varnothing$;
4: **end if**
5: $\mathcal{P} =$ sortByScore($\mathcal{P}$);
6: MaxPriorityQueue q = new Queue();
7: Set solution = new Set();
8: q.add($\varnothing$, o); // add $\varnothing$ with priority o
9: **while** ¬q.isEmpty() **do**
10: $P' =$ q.getFirst();
11: q.removeFirst();
12: $P = \rho(P')$;
13: **for all** $p \in P$ **do**
14: $\sigma = score(p)$;
15: **if** $\sigma == 1$ **then**
16: solution.add(p);
17: **else**
18: q.add(p, σ);
19: **end if**
20: **end for**
21: **end while**
22: **return** solution;

17.4.4 *Search Strategy*

As already mentioned in (Pernelle et al., 2013), the number of nodes to visit in the key discovery problem is exponential w.r.t. the number of properties considered. More precisely, given n properties, the computational complexity of our algorithm is $O(2^n)$ in the worst case, i.e. when there exists one only key formed by all properties. We tackle this issue by introducing a *fast search* strategy, which can be enabled to speed up the computation. Within this optional setting, whenever a key is found, at the next iteration all branches containing parts of the key are pruned from the refinement tree. This strategy tries to improve the runtime while fostering diversity among the discovered keys. Moreover, we consider properties whose atomic candidate keys have a score greater than a threshold τ. This lets the algorithm discard properties that alone distinguish less instances, thus having a lower probability to be part of a key.

17.5 RELATED WORK

Key discovery is a rather new research field within the domain of Linked Data, although the issue of finding keys among fields has been inherited from relational

databases. However, relational databases do not consider semantics (e.g., subsumption relations) which belong to the core of Linked Data. Previous work on key discovery for the Semantic Web can be found in (Atencia et al., 2014; Pernelle et al., 2013; Symeonidou et al., 2014). For instance, KD2R is an automatic discovery tool for composite keys in RDF data sources that may conform to different schemata (Pernelle et al., 2013). It relies on the creation of prefix trees, which serve for finding maximal undetermined keys and non-keys. However, state-of-the-art approaches as Linkkey and SAKey have shown to outperform KD2R on runtime and number of generated keys (Atencia et al., 2014; Symeonidou et al., 2014). To the best of our knowledge, not only is ROCKER the first refinement-operator-based approach for key discovery, it is also the first machine-learning-based approach for key discovery.

Independently on the application domain, the key discovery problem is a subproblem of Functional Dependencies (FDs), as every element is distinguishable only by its attributes. Keys or FDs are widely used in ontology alignment, as well as in data mining (Mannila and Toivonen, 1997), reverse engineering (Chiang et al., 1994), and query optimization (Ilyas et al., 2004; Mannila and Räihä, 1994). In particular, blocking methods such as (Michelson and Knoblock, 2006) utilize approximate keys to reduce the computational complexity of dataset joins. Unsupervised learning approaches aim at finding links among datasets by comparing datatype values of properties contained into minimal keys (Song and Heflin, 2011). The so-called collective or global approaches of data linking use keys to generate identity links between instance joins for the final scope of enriching the ontology with the collected information (Saïs et al., 2009; Arasu et al., 2009).

As previously mentioned, one of the main application areas of ROCKER is link discovery. Several approaches have been developed in previous works to detect matching properties and using them for link discovery. For example, (Ngonga Ngomo et al., 2011) relies on the hospital-residents problem to detect property matches. Other approaches based on genetic programming (e.g., (Nikolov et al., 2012)) detect matching properties while learning link specifications, which currently implements several time-efficient approaches for link discovery. (Ngonga Ngomo and Auer, 2011) proposes an approach based on the Cauchy-Schwarz inequality that allows discarding a large number of superfluous comparisons. HYPPO (Ngonga Ngomo, 2012b) and $\mathcal{HR}^3$ (Ngonga Ngomo, 2012a) rely on space tiling in spaces with measures that can be split into independent measures across the dimensions of the problem at hand. In particular, $\mathcal{HR}^3$ was shown to be the first approach that can achieve a relative reduction ratio r' less or equal to any given relative reduction ratio $r > 1$. In the ACIDS approach, similarity measures are performed on property values in order to yield features for machine-learning classifiers as support vectors machines (Soru and Ngonga Ngomo, 2012). Amongst other link discovery approaches, RDF-AI (Scharffe et al., 2009) relies on a five-step method that comprises the preprocessing, matching, fusion, interlink and post-processing of datasets.

17.6 EVALUATION

17.6.1 *Experimental Setup*

We evaluated ROCKER w.r.t. four characteristics: its runtime, RAM consumption, key extraction quality, and reduction ratio RR (Pernelle et al., 2013) between visited and total nodes.

$$RR(\alpha) = 1 - \frac{|vnodes(\alpha)|}{2^{|\mathcal{P}|}}. \tag{17.10}$$

Our approach was evaluated on data from twelve different datasets. The first two datasets were chosen in order to evaluate ROCKER on an existing artificial benchmark. Both `Restaurant 1` and `2` belong to the Ontology Alignment Evaluation Initiative (OAEI) benchmark. We then evaluated the scalability of ROCKER on ten other datasets generated from DBpedia. We built the datasets using the RDFSlice tool (Marx et al., 2013), so that each of them contains a class with its instances and their CBD. According to DBtrends[4], these classes rank among the top 20 of the most populated classes in DBpedia 3.9. The domains vary from geography (`Village`, `ArchitecturalStructure`) to professionals (`Artist`, `SoccerPlayer`) and abstract concepts (`PersonFunction`, `CareerStation`).

The generation of new evaluation datasets was preferred over the use of existing datasets due to the following reasons:

1. Datasets from the current state-of-the-art approaches contain a maximum of 1.6M triples, while ours scale up to 17.1M triples.
2. Some of the existing datasets were not formatted properly.
3. To the best of our knowledge, no key discovery benchmark has been created to date.

The lack of a manually-annotated gold standard for key discovery did not only affect the choice of the datasets. This led us to adopt the number of retrieved keys and the precision to measure the key extraction quality. In fact, while calculating the precision of a key discovery algorithm by annotating the retrieved keys is a feasible task, the set of all minimal keys needs to be known in order to compute the recall.

We compared ROCKER against two state-of-the-art approaches dubbed Linkkey (Atencia et al., 2014) and SAKey (Symeonidou et al., 2014). While Linkkey is a tool able to retrieve keys, SAKey is more scalable and able to retrieve also k-almost keys (see Section 17.4.2).

ROCKER was implemented in Java as part of the link discovery framework LIMES.[5] The datasets and the algorithm source code are also available online.[6] We launched ROCKER with two different settings; the former aims at finding minimal keys ($\alpha = 1$), while the latter aims at finding minimal almost-keys ($\alpha < 1$). Both settings were set to use the *fast search* option with $\tau = 0.001$. For the sake of simplicity, we assigned the same value to α (0.999) for all datasets when retrieving almost-keys. All experiments were carried out on a 64-bit Ubuntu Linux machine with 16 GB of RAM and an octa-core 2.5 GHz CPU.

4 http://dbtrends.aksw.org/
5 http://limes.sf.net
6 http://github.com/AKSW/rocker/

17.6.2 *Results*

Table 17.1 presents the results we obtained on the twelve datasets. Runtimes in milliseconds are reported for both tasks, i.e. "find minimal keys" and "find minimal almost-keys". For each dataset, the size in number of triples is also shown. As seen in table, all the computation runtimes for the artificial datasets lie within the same magnitude order of 1,000 milliseconds. Both ROCKER runs were slower than the other approaches, however this trend has been disproved by the following results. On the medium-sized datasets `PersonFunction`, `CareerStation` and `OrganisationMember`, our approach is the only one which completed all three tasks. In particular, Linkkey reached the Java heap space on the first two, while SAKey did not complete on the third one. On the seven remaining datasets whose size in NTriples format is larger than 1.5 GB, only our approach was able to finish the computation. This fact leads to consider ROCKER as the most scalable approach for key discovery at the state of the art.

As can be read in (Atencia et al., 2014), Linkkey was evaluated on datasets smaller than all the DBpedia datasets we generated. We thus integrated the evaluation carried out by Linkkey's authors by running the tool on our new datasets. At the same time, the largest dataset SAKey was evaluated on is comparable with our medium-sized datasets (Symeonidou et al., 2014). Results shown in Table 17.1 are thus compatible with the evaluations performed by the respective state-of-the-art algorithms.

Dataset	Triples	ROCKER(1.0)	ROCKER(0.999)	Linkkey	SAKey
OAEI 2011 Restaurant 1	1.1K	1,880	2,170	1,698	1,028
OAEI 2011 Restaurant 2	7.5K	2,424	2,833	2,278	885
DBpedia PersonFunction	383K	14,565	11,626	OutOfMemory	6,221
DBpedia CareerStation	3.0M	79,964	118,632	OutOfMemory	2,199,854
DBpedia OrganisationMember	3.9M	1,075,679	1,130,640	227,336	OutOfMemory
DBpedia Album	11.4M	1,948,767	366,147	OutOfMemory	OutOfMemory
DBpedia Artist	12.0M	203,764	168,049	OutOfMemory	OutOfMemory
DBpedia Village	12.9M	4,224,338	18,872,456	OutOfMemory	OutOfMemory
DBpedia Animal	13.7M	8,565,772	3,426,372	OutOfMemory	OutOfMemory
DBpedia SoccerPlayer	13.9M	314,853	317,285	OutOfMemory	OutOfMemory
DBpedia ArchitecturalStructure	13.3M	541,054	1,010,347	OutOfMemory	OutOfMemory
DBpedia MusicalWork	17.1M	2,524,120	2,634,869	OutOfMemory	OutOfMemory

Table 17.1: Runtime results in milliseconds for ROCKER, Linkkey and SAKey on all datasets

The node reduction ratio (RR) is shown in Table 17.2. RR expresses the rate of the number of nodes that were discarded by pruning subtrees, thus avoiding to compute their scores. The number of properties (i.e., the size of $\mathcal{P}$) and the number of visited nodes are also reported.

Table 17.3 reports the key extraction quality results. For each dataset we show the number of outcomes and the percentage of keys and minimal keys among them (i.e., precision). Many datasets have been omitted as no keys were found by any approach, or simply because the approach failed during the discovery (cf. Table 17.1). The most interesting results appear on the two OAEI datasets, where

Dataset	# properties	vnodes(1.0)	vnodes(0.999)	RR(1.0)	RR(0.999)
OAEI 2011 Restaurant 1	4	6	6	60.00%	60.00%
OAEI 2011 Restaurant 2	4	6	6	60.00%	60.00%
DBpedia PersonFunction	2	3	3	0.00%	0.00%
DBpedia CareerStation	3	4	4	42.86%	42.86%
DBpedia Organisation-Member	20	378	378	99.96%	99.96%
DBpedia Album	103	753	753	~100.00%	~100.00%
DBpedia Artist	205	928	928	~100.00%	~100.00%
DBpedia Village	116	1387	1700	~100.00%	~100.00%
DBpedia Animal	131	1188	1188	~100.00%	~100.00%
DBpedia SoccerPlayer	88	528	528	~100.00%	~100.00%
DBpedia Architectural-Structure	698	1622	3693	~100.00%	~100.00%
DBpedia MusicalWork	136	1201	1201	~100.00%	~100.00%

Table 17.2: Reduction ratios for the two settings of ROCKER on all datasets.

Linkkey was not able to recognise any key. On the other hand, SAKey was able to recognise all 3 minimal keys on `Restaurant 2`, yet it returned also 4 non-keys. SAKey was also able to find 2 out of 3 minimal keys, 3 non-minimal keys and 3 non-keys on `Restaurant 1`. Among the other datasets, 3 keys were found on `Village` by ROCKER only.

Dataset	ROCKER	Linkkey	SAKey
Restaurant 1	3 (100%, 100%)	0 (0%, 0%)	8 (62%, 25%)
Restaurant 2	3 (100%, 100%)	0 (0%, 0%)	7 (42%, 42%)
Village	3 (100%, 100%)	-	-

Table 17.3: Key extraction quality results

Namespace prefixes and the sets of almost-keys found for the DBpedia classes `ArchitecturalStructure` resp. `Village`, using a threshold for the discriminability score $\alpha = 0.999$ are shown below. Reported are 10 almost-keys that were found on `ArchitecturalStructure` and the first 20 almost-keys that were found on `Village`. As can be seen, the algorithm found six atomic almost-keys. After these had been removed from the maximal-priority queue, the refinement operator followed a path of 109 refinements through the same branch of the refinement tree. Its climb ended on a node having a discriminability score greater than α, as well as a size of 110 properties. After removing this node and its descendants from the queue, the refinement resumed from the bottom of the graph, where ROCKER found 13 more almost-keys composed by 2 properties each. As our algorithm found 84 almost-keys for `DBpedia Village`, the big size of most of the almost-keys may be the reason for the longest computation.

17.7 DISCUSSION

As presented in the previous section, ROCKER improves the state of the art w.r.t. correctness and memory consumption. Other approaches Linkkey and SAkey have

Prefix	Namespace
dbo:	http://dbpedia.org/ontology/
dbp:	http://dbpedia.org/property/
dcterms:	http://purl.org/dc/terms/
rdfs:	http://www.w3.org/2000/01/rdf-schema#
geo:	http://www.w3.org/2003/01/geo/wgs84_pos#
foaf:	http://xmlns.com/foaf/0.1/
prov:	http://www.w3.org/ns/prov#

Table 17.4: RDF prefixes used in key discovery results

shown to require much more memory than ours, as they could not return any result on bigger datasets. In particular, the heap space of 16 GB was reached on 8 and 9 DBpedia datasets, respectively. Unlike the other approaches, ROCKER managed to remain below the heap space by storing the hash index on disk. In fact, in-memory-based algorithms Linkkey and SAKey were not able to handle indexes for datasets having more than 10 million triples.

Runtime results showed that SAKey is the fastest approach on small datasets, being 1.5 to 3 times faster than the others. This could be explained by the fact that the index creation task is quicker for in-memory-based algorithms. Moreover, the outcome analysis presented in Table 17.3 confirmed that Linkkey and SAKey found candidates that obey their respective key definitions. As mentioned before, the key definition introduced in this work is more correct. A stricter definition leads ROCKER to a farther exploration of the knowledge graph, whereas the other approaches stop. Thus, the runtime is affected. Nevertheless, as can be seen, the runtime is compensated by a substantial improvement in the quality of the results.

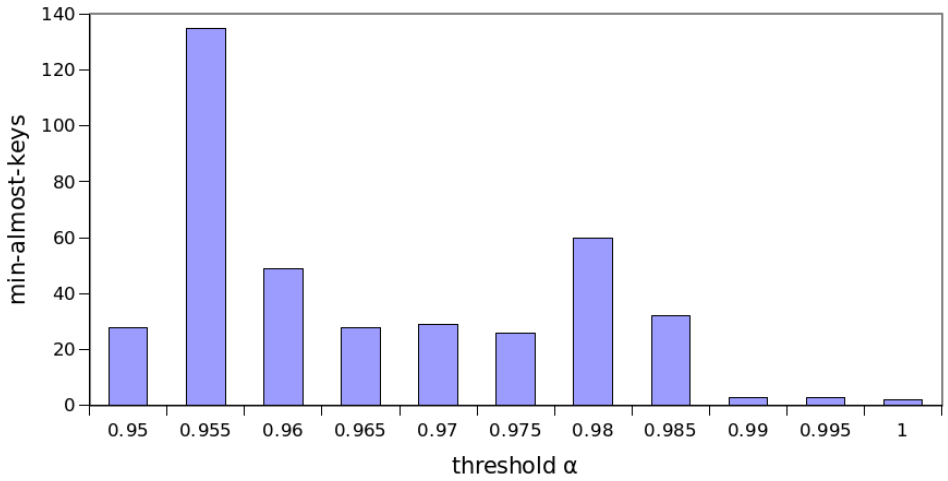


Figure 17.4: Number of minimal almost-keys found in function of threshold α for ROCKER on dataset DBpedia Monument

In order to analyse how the key discovery task varies w.r.t. the threshold α, we ran ROCKER on one chosen dataset DBpedia Monument. Figure 17.4 shows the

Size	Properties	Score
4	[foaf:name, geo:long, dbo:location, dbp:hasPhotoCollection]	0.99905
4	[foaf:name, geo:long, dbp:hasPhotoCollection, foaf:homepage]	0.99905
4	[foaf:name, geo:long, dbo:elevation, dbp:hasPhotoCollection]	0.99905
4	[foaf:name, geo:long, dbp:hasPhotoCollection, dbo:runwayLength]	0.99905
4	[foaf:name, geo:long, dbo:openingYear, dbp:hasPhotoCollection]	0.99905
4	[dbo:height, foaf:name, geo:long, dbp:hasPhotoCollection]	0.99905
4	[dbo:river, foaf:name, geo:long, dbp:hasPhotoCollection]	0.99905
4	[dbo:buildingStartYear, foaf:name, geo:long, dbp:hasPhotoCollection]	0.99905
4	[foaf:name, geo:long, dbp:hasPhotoCollection, dbo:part]	0.99905
4	[foaf:name, geo:long, dbp:hasPhotoCollection, dbo:primaryFuelType]	0.99905
1	[dbo:wikiPageID]	0.99995
1	[rdfs:label]	0.99995
1	[prov:wasDerivedFrom]	0.99995
1	[dbp:hasPhotoCollection]	0.99995
1	[foaf:isPrimaryTopicOf]	0.99995
1	[dbo:wikiPageRevisionID]	0.99995
2	[foaf:name, rdfs:comment]	0.99958
2	[geo:long, rdfs:comment]	0.99973
2	[geo:lat, rdfs:comment]	0.99968
2	[rdfs:comment, dbp:name]	0.99955
2	[dbo:wikiPageWikiLink, rdfs:comment]	0.99914
2	[rdfs:comment, dbp:wikiPageUsesTemplate]	0.99911
2	[dbo:isPartOf, rdfs:comment]	0.99901
2	[dbo:wikiPageLength, rdfs:comment]	0.99973
2	[rdfs:comment, dbo:wikiPageExternalLink]	0.99909
2	[rdfs:comment, dcterms:subject]	0.99902
2	[dbp:longd, rdfs:comment]	0.99904
2	[rdfs:comment, dbp:latd]	0.99900
2	[rdfs:comment, dbo:wikiPageOutDegree]	0.99900

Table 17.5: Examples of property sets and corresponding scores

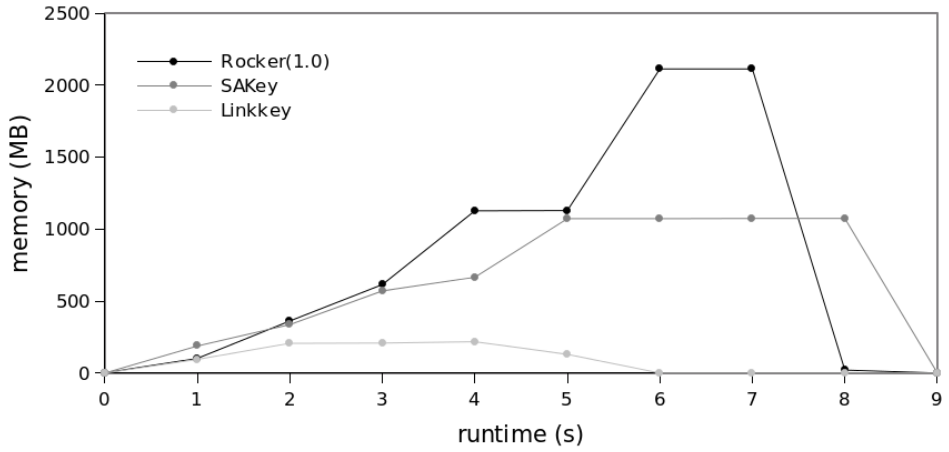


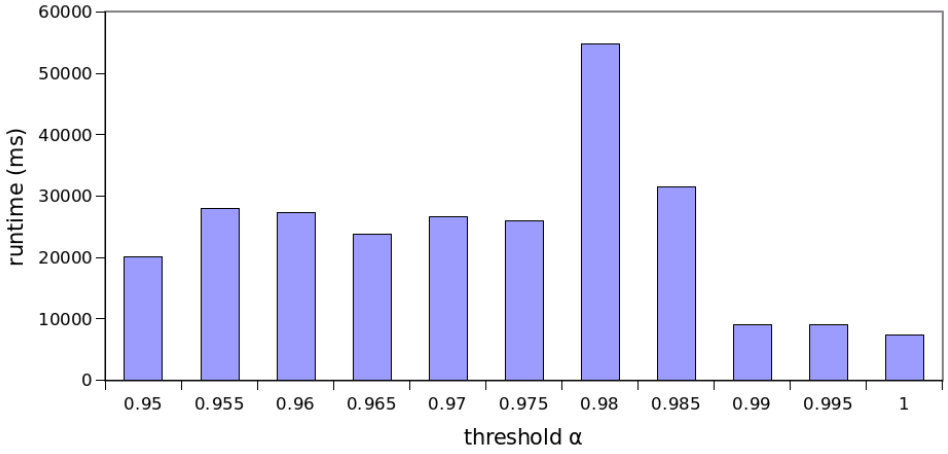
Figure 17.5: Linkkey showed the best runtime and RAM consumption performances on `DBpedia Monument`, confirming the results in Table 17.1

number of minimal almost-keys found for values of α within the interval $[0.95, 1]$ with a step of 0.005. As can be seen, values are not in scale, i.e. a minimal almost-key for α_0 does not necessarily belong to the set of minimal almost-keys for $\alpha_1 < \alpha_0$. This is because threshold α can "block" the computation before the following refinement. For instance, the highest value was reported for $\alpha = 0.955$, where 136 minimal almost-keys were found. Most of these keys are formed by a common root of two properties, which we call p_1 and p_2, in the form $\{p_1, p_2, p_i\}$ with $i = 3, \ldots, 96$. Since the discriminability score of $\{p_1, p_2\}$ is 0.953, it is not considered as minimal almost-key for $\alpha = 0.955$. However for $\alpha = 0.95$, $\{p_1, p_2\}$ will be a minimal almost-key and its descendants will not be visited, thus reducing the number of almost-keys and the runtime (see Figure 17.6a).

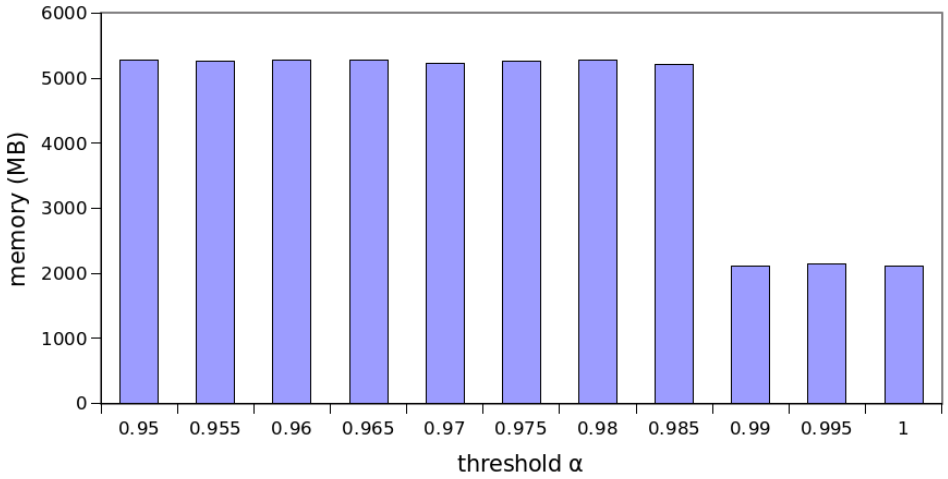
Figure 17.6b shows the memory consumption w.r.t. α. For $\alpha \geqslant 0.99$, ROCKER required less memory ($\sim$ 2 GB) than on the other experiments ($\sim$ 5.2 GB), because all the almost-keys were found before visiting the remaining refinement tree. The fact that no other almost-key exists is ensured by evaluating the score for the top element of the refinement tree, which contains all the remaining properties. Having this a score less than α, ROCKER ends the computation.

17.8 CONCLUSION AND FUTURE WORK

In this chapter, we presented the first refinement operator for key discovery. We showed that the operator is non-redundant, non-complete and finite. We implemented the operator within the ROCKER approach and showed how it can be extended to scale even on large knowledge bases. Our evaluation of ROCKER suggests that it goes beyond the state of the art with respect to its correctness and memory efficiency, while achieving comparable runtimes. Future directions include a study of the run times, number of keys and visited nodes w.r.t. the input threshold. Then, we will investigate on optimization by using in-memory storage for the hash tables, in order to decrease the query runtimes. Moreover, we will fully integrate the key discovery algorithm in LIMES and make it available in the next releases. We will then experiment with combining key discovery with the de-



(a) Runtime on Momument class



(b) Memory consumption on Momument class

Figure 17.6: Runtimes and Random Access Memory consumption as a function of the threshold α for ROCKER on the dataset `DBpedia Monument`

tection of property alignments and use those alignments within the context of link discovery.

Part IV

THE LIMES FRAMEWORK

After presenting solutions for the runtime and accuracy problems in Part II and Part III of this work, we now focus on the framework within which these solutions were implemented. With LIMES, we make a scalable framework for link discovery available that can be not only for the rapid computation of links but also for learning link specifications. In the following, we present the original architecture of the LIMES framework as well as a detailed introduction into how to use it for linking (Chapter 18). Moreover, we give a brief overview of a user interface for LIMES (Chapter 19) as well as a repository within which some of the links generated with LIMES can be found (Chapter 20).

18

ARCHITECTURE AND USE

PREAMBLE

This section describes how to use the LIMES framework through an XML- and an RDF configuration file. The section is taken from the user manual of the framework, which be found at `http://limes.sf.net`. The section and the framework were mainly written by the author in collaboration with the authors of (Hillner and Ngonga Ngomo, 2011; Ngonga Ngomo et al., 2014; Ngonga Ngomo et al., 2011; Ngonga Ngomo and Lyko, 2012, 2013; Ngonga Ngomo et al., 2013; Soru and Ngonga Ngomo, 2013; Soru et al., 2015b; Hassan et al., 2015; Ngonga Ngomo and Hassan, 2016; Georgala et al., 2016).

18.1 INTRODUCTION

LIMES, the **Li**nk Discovery Framework for **Me**tric **S**paces, is a framework for discovering links between entities contained in Linked Data sources. LIMES is a hybrid framework that combines the mathematical characteristics of metric spaces as well prefix-, suffix- and position filtering to compute pessimistic approximations of the similarity of instances. These approximations are then used to filter out a large amount of those instance pairs that do not suffice the mapping conditions. By these means, LIMES can reduce the number of comparisons needed during the mapping process by several orders of magnitude and complexity without loosing a single link.

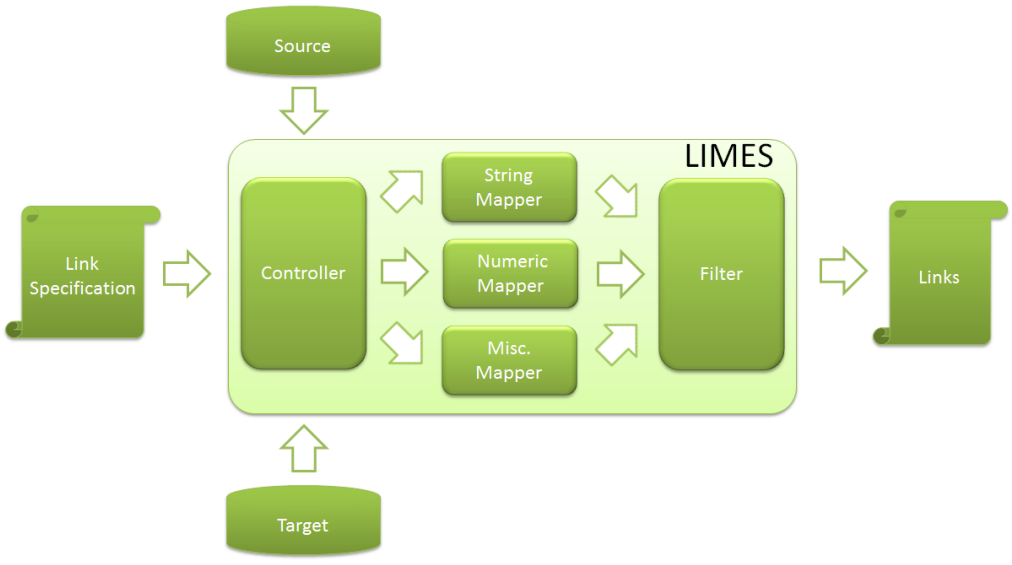


Figure 18.1: General Workflow of LIMES

The general workflow implemented by the LIMES framework is depicted in Figure 18.1. Given the source S, the target T and a link specification, LIMES first separates the different data types to merge. Strings are processed by using suffix-,

prefix- and position filtering in the string mapper. Numeric values (and all values that can be mapped efficiently to a vector space) are mapped to a metric space and processed by the HYPPO algorithm. All other values are mapped by using the miscellaneous mapper. The results of all mappers processing are filtered and merged by using time-efficient set and filtering operations.

The advantages of LIMES' approach are manifold. First, it implements **highly time-optimized** mappers, making it a complexity class faster than other Link Discovery Frameworks. Thus, the larger the problem, the faster LIMES is w.r.t. other Link Discovery Frameworks. In addition, **LIMES is guaranteed to lead to exactly the same matching as a brute force approach while at the same time reducing significantly the number of comparisons**. In addition, LIMES supports a **large number of input and output formats** and can be extended very easily to fit new algorithms , new datatypes, new preprocessing functions and others thank to its modular architecture displayed in Figure 18.2.

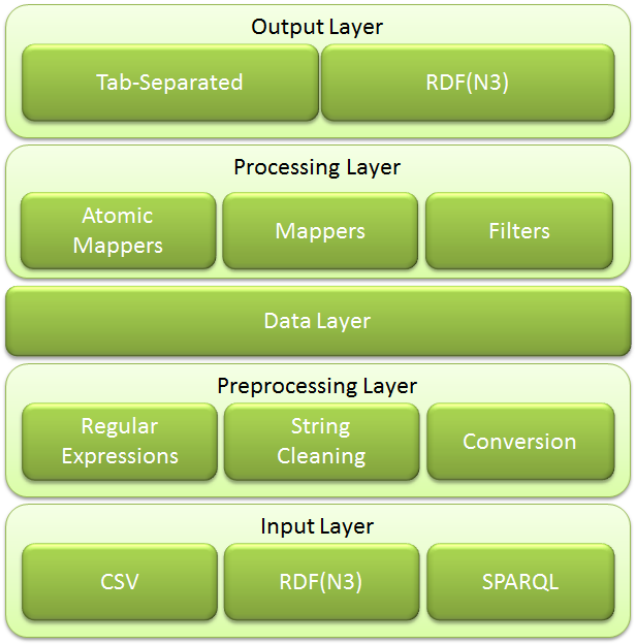


Figure 18.2: Architecture of LIMES

In general, LIMES can be used to set links between two data sources, e.g., a novel data source created by a data publisher and existing data source such as DBpedia.[1] This functionality can also be used to detect duplicates within one data source for knowledge curation. The only requirement to carry out these tasks is a simple XML-based configuration file. The purpose of this manual is to explicate the LIMES Configuration Language (LCL) that underlies these configuration files, so as allow users to generate their own configurations. An online version of LIMES is available online at `http://limes.aksw.org`.

18.2 COMPONENTS OF A LIMES CONFIGURATION FILE

A LIMES configuration file consists of ten parts, of which some are optional:

1. Metadata

1 http://dbpedia.org

2. Prefixes
3. Source data source
4. Target data source
5. Metric for similarity measurement
6. Acceptance condition
7. Review condition
8. Execution mode (optional)
9. Granularity (optional)
10. Output format

In the following, we will explicate these components by showing successively how LIMES can be configured to compute a mapping between diseases contained in Bio2RDF and LinkedCT.

18.2.1 *Metadata*

The metadata for a LIMES config file always consists of the following bits of XML:

Listing 18.1: Declaration of metadata

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE LIMES SYSTEM "limes.dtd">
<LIMES>
```

18.2.2 *Prefixes*

Defining a prefix in a LIMES file demands setting two values: the namespace that will be addresses by the prefix and the prefix per se, as shown below.

Listing 18.2: Namespace declaration

```
<PREFIX>
<NAMESPACE>http://www.w3.org/1999/02/22-rdf-syntax-ns#</NAMESPACE>
<LABEL>rdf</LABEL>
</PREFIX>
```

Here, we set the prefix `rdf` to correspond to `http://www.w3.org/1999/02/22-rdf-syntax-ns#`. A LIMES link specification can contain as many prefixes as required.

18.2.3 *Source Data Source*

LIMES computes links between items contained in two Linked Data sources dubbed source and target. An example of a configuration for a source data source is shown below.

Listing 18.3: Declaration of a source dataset

```
<SOURCE>
<ID>mesh</ID>
<ENDPOINT>http://mesh.bio2rdf.org/sparql</ENDPOINT>
<VAR>?y</VAR>
<PAGESIZE>5000</PAGESIZE>
<RESTRICTION>?y rdf:type meshr:Concept</RESTRICTION>
<PROPERTY>dc:title</PROPERTY>
<TYPE>sparql</TYPE>
</SOURCE>
```

Six properties need to be set.

1. Each data source must be given an ID via the tag `ID`.
2. The SPARQL endpoint of the data source needs to be explicated via the `ENDPOINT` tag. In case local files (CSV, N3, TURTLE, etc.) are to be linked, `ENDPOINT` should be set to the absolute path of the file containing the data to link.
3. The variable associated with this endpoint must be specified. This is done by setting the `VAR` tag. This variable is used later when specifying the metric used to compare the entities retrieved from the source and target endpoints.
4. The fourth property is set via the `PAGESIZE` tag. This property must be set to the maximal number of triples returned by the SPARQL endpoint to address. For example, the DBpedia endpoint at `http://dbpedia.org/sparql` returns a maximum of 1000 triples for each query. LIMES' SPARQL module can still retrieve all relevant instances for the mapping if given this value. If the SPARQL endpoint does not limit the number of triple it returns or if the input is a file, the value of `PAGESIZE` should be set to -1.
5. The restrictions of the data to retrieved can be set via the `RESTRICTION` tag. This tag allows to limit the entries that are retrieved the LIMES' query module. In this particular example, we only instances of MESH concepts.
6. The `PROPERTY` tag allows to specify the properties that will be used during the linking. It is important to note that the property tagcan also be used to specify the preprocessing on the input data. For example, setting `rdfs:label AS nolang`, one can ensure that the language tags get removed from each `rdfs:label` before it is written in the cache. Pre-processing functions can be piped into one another by using `->`. For example, `rdfs:label AS nolang->lowercase` will compute `lowercase(nolang(rdfs:label))`.

The pre-processing functions include:

- `nolang` for removing language tags,
- `lowercase` for converting the input string into lower case,
- `uppercase` for converting the input string into upper case,
- `number` for ensuring that only the numeric characters, "." and "," are contained in the input string,

- `replace(String a,String b)` for replacing each occurrence of `a` with `b`,
- `cleaniri` for removing all the prefixes from IRIs,
- `celsius` for converting Fahrenheit to Celsius,
- `fahrenheit` for converting Celsius to Fahrenheit.

Sometimes, generating the right link specification might either require merging property values (for example, the `dc:title` and `foaf:name` of MESH concepts) or splitting property values (for example, comparing the label and `foaf:homepage` of source instances and the `foaf:homepage` of target instances as well as `foaf:homepage AS cleaniri` of the target instances with the `rdfs:label` of target instances. To enable this goal, LIMES provides the `RENAME` operator which simply store either the values of a property or the results of a preprocessing into a different property field. For example, `foaf:homepage AS cleaniri RENAME label` would stored the homepage of a object without all the prefixes in the name property. The user could then access this value during the specification of the similarity measure for comparing sources and target instances. Note that the same property value can be used several times. Thus, the following specification fragment is valid and leads to the the `dc:title` and `foaf:name` of individuals) of MESH concepts being first cast down to the lowercase and then merged to a single property.

Listing 18.4: Declaration of prerpocessing functions

```
<SOURCE>
<ID>mesh</ID>
<ENDPOINT>http://mesh.bio2rdf.org/sparql</ENDPOINT>
<VAR>?y</VAR>
<PAGESIZE>5000</PAGESIZE>
<RESTRICTION>?y rdf:type meshr:Concept</RESTRICTION>
<PROPERTY>dc:title AS lowercase RENAME name</PROPERTY>
<PROPERTY>foaf:name AS lowercase RENAME name</PROPERTY>
<TYPE>sparql</TYPE>
</SOURCE>
```

In addition, the following allows splitting the values of `foaf:homepage` into the property values `name` and `homepage`.

Listing 18.5: Splitting properties

```
<SOURCE>
<ID>mesh</ID>
<ENDPOINT>http://mesh.bio2rdf.org/sparql</ENDPOINT>
<VAR>?y</VAR>
<PAGESIZE>5000</PAGESIZE>
<RESTRICTION>?y rdf:type meshr:Concept</RESTRICTION>
<PROPERTY>foaf:homepage AS lowercase RENAME homepage</PROPERTY>
<PROPERTY>foaf:homepage AS cleaniri->lowercase RENAME name</PROPERTY>
<TYPE>sparql</TYPE>
</SOURCE>
```

In addition, a source type can be set via `TYPE`. The default type is set to `SPARQL` (for a SPARQL endpoint) but LIMES also supports reading files directly from the harddrive. The supported data formats are

- `CSV`: Character-separated file can be loaded directly into LIMES. Note that the separation character is set to `TAB` as a default. The user can alter this setting programmatically.
- `N3` (which also reads `NT` files) reads files in the `N3` language.
- `N-TRIPLE` reads files in W3C's core N-Triples format.[2]
- `TURTLE` allows reading files in the `Turtle` syntax.[3]

Consequently, if you want to download data from a SPARQL endpoint, there is no need to set the `<TYPE>` tag. If instead you want to read the source (or target) data from a file, the `<ENDPOINT>` tag should contain the path to the file to read, e.g. `<ENDPOINT>C:/Files/dbpedia.nt</ENDPOINT>` In addition, the `<TYPE>` tag then needs to be set, for example by writing `<TYPE>NT</TYPE>`.

18.2.4 *Target Data Source*

Configuring the target data source is very similar to configuring the source data source. The only difference lies in the beginning tag, i.e., `TARGET` instead of `SOURCE`. In the example shown below, we retrieve the `condition_name` of a condition from LinkedCT. We do no set the type of the source. Thus, LIMES supposes it is a SPARQL endpoint.

Listing 18.6: Declaration of a target dataset

```
<TARGET>
<ID>linkedct</ID>
<ENDPOINT>http://data.linkedct.org/sparql</ENDPOINT>
<VAR>?x</VAR>
<PAGESIZE>5000</PAGESIZE>
<RESTRICTION>?x rdf:type linkedct:condition</RESTRICTION>
<PROPERTY>linkedct:condition_name</PROPERTY>
</TARGET>
```

18.2.5 *Metric Expression for Similarity Measurement*

One of the core improvements of the newest LIMES kernels is the provision of a highly flexible language for the specification of complex metrics for linking (set by using the `METRIC` tag as exemplified below).

Listing 18.7: Declaration of a simple similarity measure

```
<METRIC>
trigrams(y.dc:title, x.linkedct:condition_name)
</METRIC>
```

In this example, we use the Trigrams metric to compare the `dc:title` of the instances retrieved from the source data source, with which the variable `y` is associated, with the `linkedct:`

2 http://www.w3.org/TR/rdf-testcases/#ntriples
3 http://www.w3.org/TR/turtle/

condition_name of the instances retrieved from the target data source, with which the variable x is associated. While such simple metrics can be used in many cases, complex metrics are necessary in complex linking cases. LIMES includes a formal grammar for specifying complex configurations of arbitrary complexity. For this purpose, two categories of binary operations are supported: Metric operations and boolean operations.

Metric operations allow to combine metric values. They include the operators MIN, MAX, ADD and MULT, e.g. as follows:

Listing 18.8: Declaration of a complex similarity measure

```
MAX(trigrams(x.rdfs:label,y.dc:title),euclidean(x.lat|long, y.latitude|
    longitude)).
```

This specification computes the maximum of (1) the trigram similarity of x's rdfs:label and y's dc:title and (2) the 2-dimension euclidean distance of x's lat and long mit y's latitude and longitude, i.e.,
$\sqrt{(x.lat - y.latitude)^2 + (x.long - y.longitude)^2}$.

Note that euclidean supports arbitrarily many dimensions. In addition, note that ADD allows to define weighted sums as follows:

Listing 18.9: Linear combination of measures

```
ADD(0.3*trigrams(x.rdfs:label,y.dc:title),
    0.7*euclidean(x.lat|x.long, y.latitude|y.longitude)).
```

Boolean operations allow to combine and filter the results of metric operations and include AND, OR, DIFF, e.g. as follows:

Listing 18.10: Specification of complex measures through Boolean operators

```
AND(trigrams(x.rdfs:label,y.dc:title)|0.9,
    euclidean(x.lat|x.long, y.latitude|y.longitude)|0.7).
```

This specification returns all links such that (1) the trigram similarity of x's rdfs:label and y's dc:title is greater or equal to 0.9 and (2) the 2-dimension euclidean distance of x's lat and long mit y's latitude and longitude is greater or equal to 0.7.

The current version of LIMES supports the string metrics

- Trigrams,
- Cosine,
- Jaccard,
- Levenshtein,
- Jaro and
- Jaro-Winkler

Overlap as well as Monge-Elkan are currently being added. In addition it supports comparing numeric vectors by using the

- Euclidean metric as well as

- the `Orthodromic` distance.

While the Euclidean measure can deal with n-dimensional data, the orthodromic distance assumes that it is given a WKT POINT as input. If the input is a polygon, it uses the Hausdorff distance. The similarity between polygons can be measured by using the

- `Hausdorff` distance,
- `Sum of minimums` distance,
- `Fréchet` distance,
- `Fair surjection`,
- `Surjection` and
- `SymmetricHausdorff` distance.

Currently, these distances can deal with POLYGON and LINESTRING. More complex distance measures are being added.

18.2.6 *Acceptance Condition*

Setting the acceptance condition basically consists of setting the value for the threshold above which links are considered to be valid and not to required further curation. This can be carried out as exemplified below.

Listing 18.11: Declartion of acceptance threshold

```
<ACCEPTANCE>
<THRESHOLD>0.98</THRESHOLD>
<FILE>accepted.nt</FILE>
<RELATION>owl:sameAs</RELATION>
</ACCEPTANCE>
```

By using the `THRESHOLD` tag, the user can set the value for the metric value above which two instances are considered to be linked via the relation specified by using the tag `RELATION`, i.e., `owl:sameAs` in our example. Setting the `FILE` allows to specify where the links should be written. Currently, LIMES produces output files in the N3 format.

Future versions of LIMES will allow to write the output to other streams and in other data formats.

18.2.7 *Review Condition*

Setting the condition upon which links must be reviewed manually is very similar to setting the acceptance condition as shown below.

Listing 18.12: Declaration of verification threshold

```
<REVIEW>
<THRESHOLD>0.95</THRESHOLD>
<FILE>reviewme.nt</FILE>
<RELATION>owl:sameAs</RELATION>
</REVIEW>
```

All instances that have a similarity between the threshold set in `REVIEW` (0.95 in our example) and the threshold set in `ACCEPTANCE` (0.98 in our example) will be written in the review file and linked via the relation set in `REVIEW`.

The LIMES configuration file should be concluded with `</LIMES>`

18.2.8 *Execution Mode (optional)*

The user can choose between the executions modes `SIMPLE` and `FILTER` to tune LIMES' runtime.

Listing 18.13: Declaration of execution modes

```
<EXECUTION>SIMPLE</EXECUTION>
```

Moreover, the user can select how the mappings returned by LIMES are to be postprocessed. `OneToN` leads to LIMES returning only the best matching t to any given s in the mapping $M = \{(s,t) \in S \times T\}$.`OneToOne` leads to LIMES aiming to find the best one-to-one mapping out of the output in a way similar to that above.

18.2.9 *Granularity (optional)*

The user can choose positive integers to set the granularity of HYPPO, HR3 or ORCHID by setting

```
<GRANULARITY>2</GRANULARITY>.
```

18.2.10 *Output Format*

The user can choose between TAB and N3 as output format by setting

```
<OUTPUT>N3</OUTPUT>
```

18.3 EXAMPLE OF A CONFIGURATION FILE

The following shows the whole configuration file for LIMES explicated in the sections above.

Listing 18.14: Example of an XML Specification

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE LIMES SYSTEM "limes.dtd">
<LIMES>
<PREFIX>
        <NAMESPACE>http://www.w3.org/1999/02/22-rdf-syntax-ns#</NAMESPACE>
        <LABEL>rdf</LABEL></PREFIX>
<PREFIX>
        <NAMESPACE>http://www.w3.org/2000/01/rdf-schema#</NAMESPACE>
        <LABEL>rdfs</LABEL></PREFIX>
<PREFIX>
        <NAMESPACE>http://www.w3.org/2002/07/owl#</NAMESPACE>
        <LABEL>owl</LABEL></PREFIX>
<PREFIX>
```

```
        <NAMESPACE>http://data.linkedct.org/resource/linkedct/</NAMESPACE>
        <LABEL>linkedct</LABEL></PREFIX>
<PREFIX>
        <NAMESPACE>http://purl.org/dc/elements/1.1/</NAMESPACE>
        <LABEL>dc</LABEL></PREFIX>
<PREFIX>
        <NAMESPACE>http://bio2rdf.org/ns/mesh#</NAMESPACE>
        <LABEL>meshr</LABEL></PREFIX>

<SOURCE>
        <ID>mesh</ID>
        <ENDPOINT>http://mesh.bio2rdf.org/sparql</ENDPOINT>
        <VAR>?y</VAR>
        <PAGESIZE>5000</PAGESIZE>
        <RESTRICTION>?y rdf:type meshr:Concept</RESTRICTION>
        <PROPERTY>dc:title</PROPERTY>
</SOURCE>

<TARGET>
        <ID>linkedct</ID>
        <ENDPOINT>http://data.linkedct.org/sparql</ENDPOINT>
        <VAR>?x</VAR>
        <PAGESIZE>5000</PAGESIZE>
        <RESTRICTION>?x rdf:type linkedct:condition</RESTRICTION>
        <PROPERTY>linkedct:condition_name</PROPERTY>
</TARGET>

<METRIC>
 MAX(trigrams(y.dc:title, x.linkedct:condition_name),
     cosine(y.dc:title, x.linkedct:name))
</METRIC>

<ACCEPTANCE>
        <THRESHOLD>0.98</THRESHOLD>
        <FILE>accepted.txt</FILE>
        <RELATION>owl:sameAs</RELATION>
</ACCEPTANCE>

<REVIEW>
        <THRESHOLD>0.95</THRESHOLD>
        <FILE>reviewme.txt</FILE>
        <RELATION>owl:sameAs</RELATION>
</REVIEW>
</LIMES>
```

LIMES can be also configured using a RDF configuration file, the next listing represent the same LIMES configuration used in the previous XML file.

Listing 18.15: Example of an RDF Specification

```
@prefix dc:      <http://purl.org/dc/elements/1.1/> .
@prefix rdfs:    <http://www.w3.org/2000/01/rdf-schema#> .
@prefix meshr:   <http://bio2rdf.org/ns/mesh#> .
@prefix linkedct:  <http://data.linkedct.org/resource/linkedct/> .
@prefix owl:     <http://www.w3.org/2002/07/owl#> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix limes:   <http://limes.sf.net/ontology/> .
```

```
8
9  limes:meshToLinkedct
10       a                    limes:LimesSpecs ;
11       limes:hasSource      limes:meshToLinkedctSource ;
12       limes:hasTarget      limes:meshToLinkedctTarget ;
13       limes:hasAcceptance  limes:meshToLinkedctAcceptance ;
14       limes:hasMetric      limes:meshToLinkedctMetric ;
15       limes:hasReview      limes:meshToLinkedctReview .
16 limes:meshToLinkedctSource
17       a                    limes:SourceDataset ;
18       rdfs:label           "mesh" ;
19       limes:endPoint       "http://mesh.bio2rdf.org/sparql" ;
20       limes:variable       "?y" ;
21       limes:pageSize       "5000" ;
22       limes:restriction    "?y rdf:type meshr:Concept" ;
23       limes:property       "dc:title" .
24 limes:meshToLinkedctTarget
25       a                    limes:TargetDataset ;
26       rdfs:label           "linkedct" ;
27       limes:endPoint       "http://data.linkedct.org/sparql" ;
28       limes:variable       "?x" ;
29       limes:pageSize       "5000" ;
30       limes:restriction    "?x rdf:type linkedct:condition" ;
31       limes:property       "linkedct:condition_name" .
32 limes:meshToLinkedctMetric
33       a                    limes:Metric ;
34       limes:expression
35          "MAX(trigrams(y.dc:title,x.linkedct:condition_name),cosine(y.dc:title
               ,x.linkedct:name))" .
36 limes:meshToLinkedctAcceptance
37       a                    limes:Acceptance ;
38       limes:threshold      "0.98" ;
39       limes:file           "accepted.txt" ;
40       limes:relation       "owl:sameAs" .
41 limes:meshToLinkedctReview
42       a                    limes:Review ;
43       limes:threshold      "0.95" ;
44       limes:file           "reviewme.txt" ;
45       limes:relation       "owl:sameAs" .
```

18.4 THE LIMES DISTRIBUTION

18.4.1 *Content*

The LIMES distribution in its current version 0.5.RC1 contains the files

- `LIMES.jar`, which implements our framework,
- `limes.dtd`, the data type definition for LIMES configuration files and
- `user_manual.pdf`, this file.

In addition, it contains the folders

- `lib`, which contains all the libraries necessary to run our framework and
- `examples`, which contains examples of configuration files.

18.4.2 *Running the Framework*

Once the configuration file (dubbed `config.xml` in this manual) has been written, the last step consists of actually running the LIMES framework. For this purpose, simply run

`java -jar LIMES.jar config.xml`.

In case your system runs out of memory, please use the `-Xmx` option to allocate more memory to the Java Virtual Machine. Please ensure that the Data Type Definition file for LIMES, `limes.dtd`, is in the same folder as the `LIMES.jar` and everything should run just fine. Enjoy.

18.5 LICENSE AND WARRANTY INFORMATION

LIMES is free to use for non-commercial purposes. For any kind of commercial use, the author is to be contacted. LIMES is distributed without any warranty of any type.

19 SAIM: A USER INTERFACE FOR LIMES

PREAMBLE

This chapter presents SAIM—a user interface that aims to support users during the creation of high-quality link specifications. The tool implements a simple but effective workflow to creating initial link specifications. In addition, SAIM allows accessing the machine learning algorithms implemented in LIMES. The content of this chapter was presented in the demo paper (Lyko et al., 2013). The author co-designed the interface, which relies heavily on the LIMES code which was mainly written by the author. Moreover, he co-wrote the paper and supervised the rest of the work.

19.1 INTRODUCTION

Links between instances are of central importance for a large number of tasks such as data integration, federated querying and knowledge retrieval as often pointed out in the literature (Auer et al., 2013a). Two main problems arise when trying to discover links between datasets or deduplicate datasets. First, naive solutions to Link Discovery (LD) need to compare all resources in the source dataset with all resources in the target dataset and, thus, have quadratic time complexity. Consequently, naive approaches are impractical when computing links across large datasets such as DBpedia[1] or Yago.[2] Time-efficient algorithms and frameworks such as LIMES (Ngonga Ngomo and Auer, 2011) and SILK[3] have been developed to reduce the number of comparisons which need to be made between resources. While these approaches achieve practicable runtimes even on large datasets, they do not guarantee the quality of the links that are returned by LD frameworks. Addressing this second problem of LD demands the development of techniques that can compute accurate *link specifications* for deciding whether two resources should be linked. Both supervised (e.g., (Ngonga Ngomo and Lyko, 2012; Ngonga Ngomo et al., 2013)) and unsupervised machine-learning approaches (e.g., (Nikolov et al., 2012)) have been proposed to achieve this goal.

SAIM[4] encompasses solutions for both problems within a simple interface which implements a flexible workflow. The tool relies on algorithms implemented in LIMES[5], which have been shown to outperform the state of the art in previous work w.r.t. time efficiency (Ngonga Ngomo, 2012b). In addition to allowing expert users to create specifications manually, SAIM implements supervised and unsupervised learning algorithms including extensions of EAGLE (Ngonga Ngomo and Lyko, 2012) and the novel COALA (Ngonga Ngomo et al., 2013) approach (presented at the same conference), which have been shown to lead to high-quality

1 http://dbpedia.org
2 http://www.mpi-inf.mpg.de/yago-naga/yago/
3 https://www.assembla.com/spaces/silk/
4 SAIM stands for (Semi-)Automatic Instance Matcher and is pronounced like "same". All information to the project including a demo and a screencast can be found at http://aksw.org/projects/saim.
5 See http://limes.sf.net.

specifications. By these means, SAIM can support domain experts and lay users during the creation of link specifications. Moreover, it implements the time-efficient algorithms for class and property matching algorithms proposed in (Ngonga Ngomo et al., 2011). SAIM goes beyond existing interfaces for link discovery (e.g., SILK Workbench[6]) by supporting several self-configuration algorithms that allow the automatic creation of link specifications. In the following, we present the workflow underlying SAIM and then focus on the content of the SAIM demonstration.

19.2 SAIM

The workflow underlying SAIM consists of four main steps: (1) data selection, (2) schema matching, (3) creation of the specification and (4) execution of the specification. In the following, we explain how SAIM supports each of these steps.

19.2.1 *Data Selection*

SAIM allows users to specify SPARQL endpoints or local RDF files (for users with logins) as data sources. SPARQL endpoints are specified by stating the URL of the endpoint and (if necessary) the graph from which the data is to read. Moreover, each endpoint can be given a name. Our software signalizes to its user whether the endpoint he selected is alive by issuing a simple SPARQL query to the specified endpoint. Moreover, it provides a list of commonly used endpoints such as DBpedia.[7] Local RDF files can be in any of the serialization formats supported by the Jena Framework [8] on which SAIM relies. In addition, the tool supports using data stored as CSV files. In the latter case, the data is converted to RDF on the fly by using the strings contained in each column of the first row as property labels and the elements of the first column as URI for the resources.

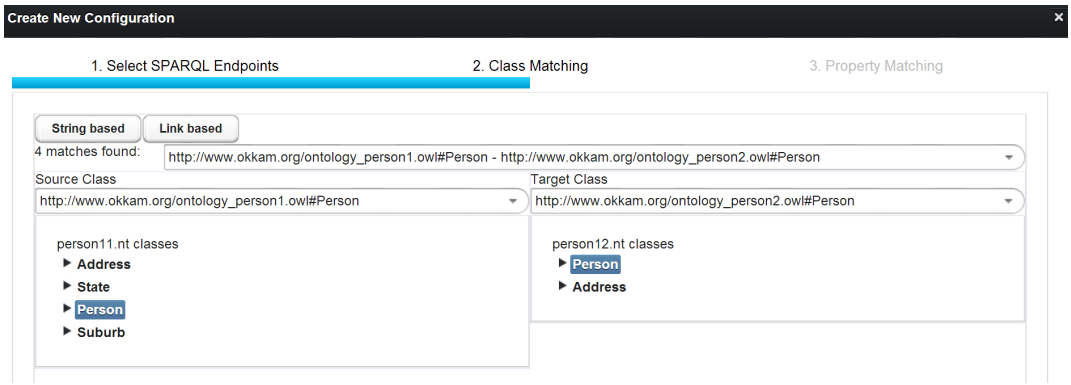


Figure 19.1: Schema Matching step in SAIM

19.2.2 *Schema Matching*

Our approach relies on simple yet the time-efficient schema matching algorithms presented for matching classes and properties (see Figure 19.1). The user can

6 https://www.assembla.com/spaces/silk/wiki/Silk_Workbench
7 http://dbpedia.org/sparql
8 http://jena.apache.org/

choose between intensional matching (`String based` button) and a matching approach based on stable marriage on links (`Link based` button, see (Ngonga Ngomo et al., 2011) for more details). Per default, SAIM compute the string-based matching between the class labels by using the trigram similarity and return a sorted list of matching classes. We chose this approach because of its time-efficiency. The user can either choose the stable-marriage-based approach or navigate through the schemas of the dataset to perform the schema matching manually. Note that SAIM implements fallback solutions to be able to display the schemas of datasets with incomplete ontologies. For example, if no statement of the form `?x a rdf:Class` is found in the dataset, our approach falls back to retrieving all distinct `?x` such that triples of the form `?y a ?x` is contained in the dataset.

19.2.3 *Creation of Specifications*

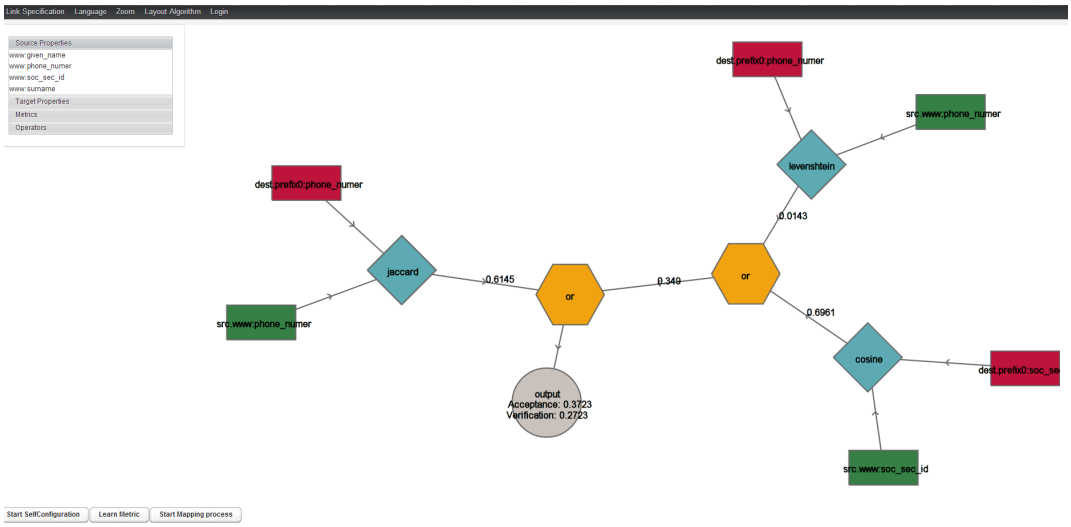


Figure 19.2: SAIM specification window

The creation of specifications is the most involved part of SAIM's workflow. Once the schema matching has been carried out, the user is presented with SAIM's main window. Initially, this window contains an *output node*. On the left, the expert user can choose between the different properties, several similarity and distance measures (incl. Levenshtein, Trigrams, Cosine) as well as operators (incl. AND, OR, MAX, MIN) to combine these metrics to a single specification manually (see Figure 19.2). The user can also choose the `Learn metric` button instead, which allows the user to select between several machine-learning algorithms for link specification learning including EAGLE (Ngonga Ngomo and Lyko, 2012) and its extensions in COALA (Ngonga Ngomo et al., 2013) (Figure 19.3). After choosing these algorithms, the user is presented the most informative positive and negative examples and can choose whether they are matches or non-matches. SAIM supports the user in this process by allowing him to view the values of the properties of the resources that are part of the match or to dereference their URIs. SAIM also enables lay users to create link specification by offering a *self-configuration* mode. In the corresponding window (see Figure 19.4), SAIM allows the user to choose which algorithm to use and whether the specification should be tuned towards precision or recall (the default setting being that both are equally important). Once

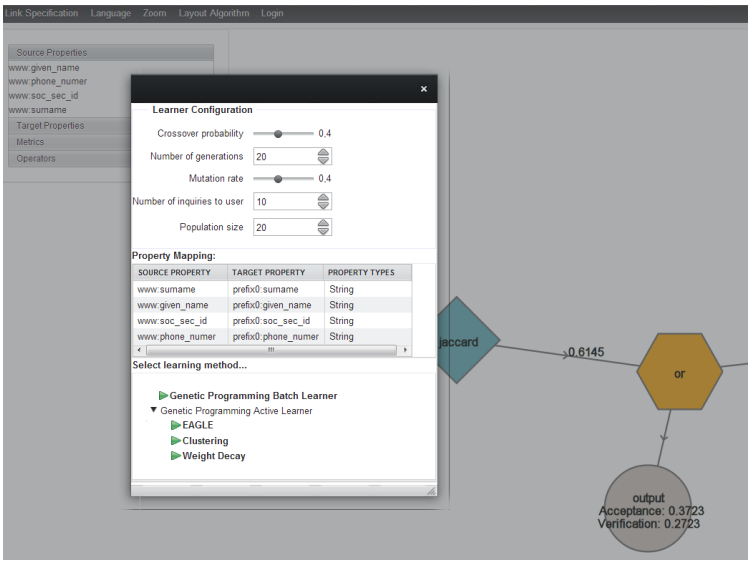


Figure 19.3: Selection of algorithms

the user has selected a configuration, SAIM runs unsupervised machine-learning algorithms based on EAGLE or RAVEN and returns the specification that maximize a selected pseudo-F-measure. The specification shown in Figure 19.2 was learned fully automatically by the unsupervised version EAGLE.

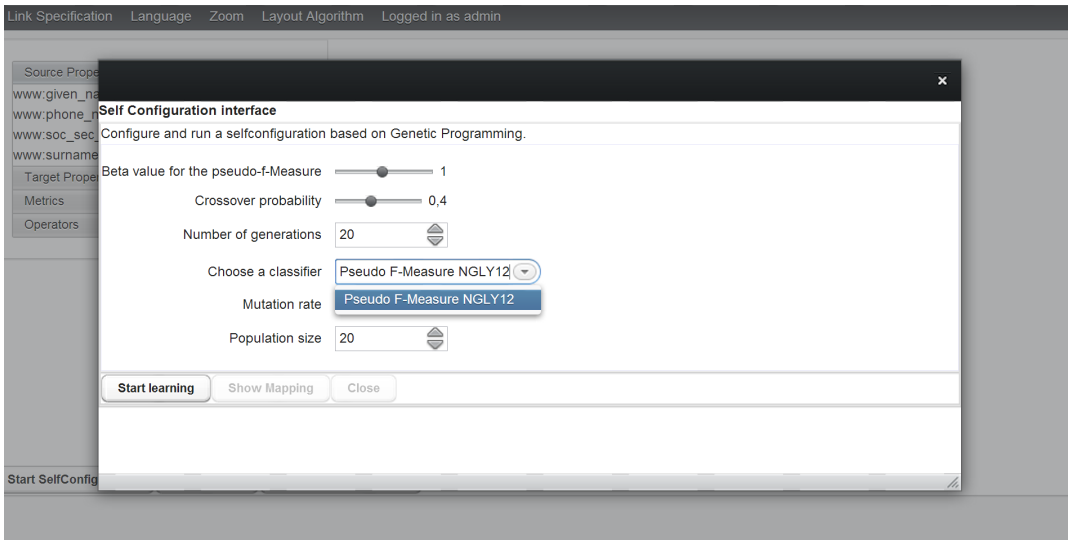


Figure 19.4: Specification of self-configuration in SAIM

19.3 DEMONSTRATION

The goal of the demonstration was to show the whole workflow described above from the datasets to the export of the resulting specification and links. We begun by showing how SAIM deals with data in SPARQL endpoints and with local datasets. Thereafter, we presented and explained how SAIM suggests class and property matchings to the user.We explained the hierarchy of fallback solutions that SAIM employs to generate both an overview of the class structure and the corresponding

class matching as well as how it uses extensional and intensional schema matching approaches for both class and property matching. In a third step, we showcased the approaches implemented in SAIM. We began by showing how the expert user can use SAIM to create link specifications manually. Thereafter, we presented how domain experts can employ the active learning algorithms EAGLE and COALA to learn and refine link specifications iteratively. Then, we showed how lay users can use unsupervised machine learning to have SAIM detect a high-quality link specification for them. Finally, we demonstrated how the results of the specification process (i.e., the link specification and the resulting mappings) can be downloaded from SAIM for use in further applications. Throughout the demonstration, we employed benchmark datasets such as those provided by the OAEI challenges.[9]

19.4 CONCLUSIONS AND FUTURE WORK

This chapter gives an overview of SAIM, an interface for the creation of high-quality link specifications. SAIM represents a further step towards the vision of zero-configuration link discovery as it allows users to create such specifications with minimal effort. In future work, we will extend SAIM with more algorithms for learning link specifications and aim to achieve our vision of easy and effective link discovery.

9 http://oaei.ontologymatching.org/

20 LINKLION: A PORTAL FOR LINKS

PREAMBLE

One of the practical application developed within the context of the work underlying this thesis is a portal for links dubbed LinkLion. Currently, the portal contains 12.6 million links of 10 different types distributed across 3,184 mappings that link 449 datasets. The content of this chapter is taken from the corresponding demo paper (Nentwig et al., 2014), which was co-written by the author, who also partook in the conception and the supervision of the implementation of the portal.

20.1 INTRODUCTION

In addition to being central for question answering across several datasets, links also play a key role in various other domains such as data fusion and federated SPARQL queries. It is a well-known problem that links make up less than 3% of the RDF triples on the Web of Data (Ngonga Ngomo and Auer, 2011). This problem is being addressed by link discovery and ontology matching tools and frameworks (Ngonga Ngomo, 2012a; Kirsten et al., 2011). However, due to the architectural choices behind the Web of Data, the results of a link discovery (LD) framework cannot be added directly to the datasets involved in the link discovery process. Further, the direct addition of links to a knowledge base fails to provide means to track the source of these links for later reference. Moreover, the availability of some endpoints still remains a major issue,[1] making the direct addition of linking results to some endpoints unattractive.

We address these drawbacks by presenting the open-source link repository LinkLion. The main goal of LinkLion is to facilitate the publication, retrieval and use of links between knowledge bases. Our repository thus provides dedicated functionality for the upload, storage, querying and download of large sets of links. Currently, it contains 63 million triples which describe 12.6 million links of 10 different types (e.g., `owl:sameAs`, `dbo:spokenIn`, `foaf:made`, `spatial:P`)[2] distributed across 3184 mappings that link 449 datasets. These links were retrieved from the Web as well as computed by tools such as LIMES (Ngonga Ngomo, 2012a) and SILK (Volz et al., 2009). Our repository provides a SPARQL query interface as well as commodity interfaces to access the mappings. In contrast to other portals such as BioPortal[3], LinkLion focuses exclusively on links and provides dedicated functionality for manipulating them. Moreover, we do not limit ourselves to a single domain such as the life sciences. In the following, we give a brief overview of the repository and show the use cases that will be presented during the demo. The repository can be accessed at `http://www.linklion.org`. The code of the repository is available at `http://github.com/AKSW/LinkingLodPortal`. The SPARQL endpoint can be found at `http://www.linklion.org:8890/sparql`.

1 `http://labs.mondeca.com/sparqlEndpointsStatus.html`
2 We used the prefixes available at `http://prefix.cc`.
3 `http://www.bioontology.org/BioPortal`

20.2 IMPLEMENTATION

An overview of LINKLION's architecture is given in Figure 20.2. The back end consists of a triple store in which we stored data according to the vocabulary shown in Figure 20.1.

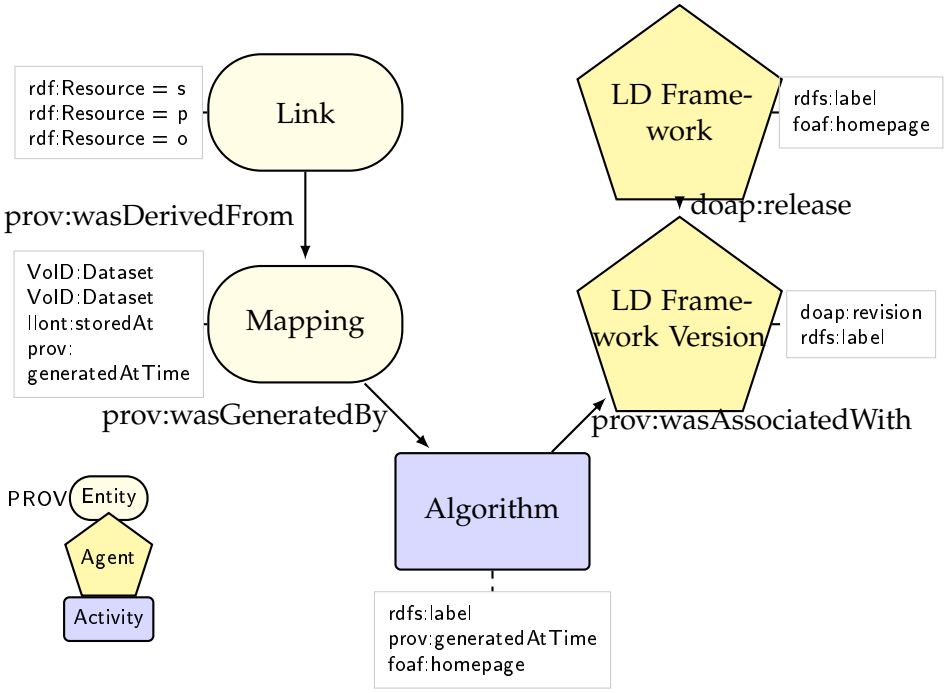


Figure 20.1: Overview of the LINKLION ontology. New classes such as Link, Mapping, Algorithm and LD Framework are specified as subclasses of the PROV vocabulary.

The ontology[4] was designed with usability and reuse in mind. Especially, we wanted to allow end users of the portal to selected dedicated portions of certain mappings at will. This meant designing an ontology that allowed amongst others (1) retrieving all links that pertain to a particular resource or set of resources, (2) gathering all mappings between datasets of interest as well as (3) getting aggregated information on how particular links came about. We implemented this vision by storing the output of a link discovery tool under an instance of the mapping class. Single mappings can be described by metadata including the datasets that they link, the tool (incl. a version number) used to generate the links and the creation date of the mapping. We refrained from using blank nodes for links. Instead, we gave each link a unique ID. Note that we reused existing vocabularies (especially PROV[5], VoID[6] and DOAP[7]) as much as we could. The use of a triple store pays off as end users can choose to provide more metadata such as the link specification used or parameters of the algorithm they used to discover the link without us having to alter our schema. For the sake of scalability, we yet also provide the core of the data in the triple store as SQL dump. The functionality of the

4 Available at http://www.linklion.org/ontology.
5 http://www.w3.org/TR/prov-o/
6 http://www.w3.org/TR/void/
7 https://github.com/edumbill/doap/

back end is exposed by RESTful interfaces, which allow a programmatic access to LinkLion from code written in virtually any modern programming language.

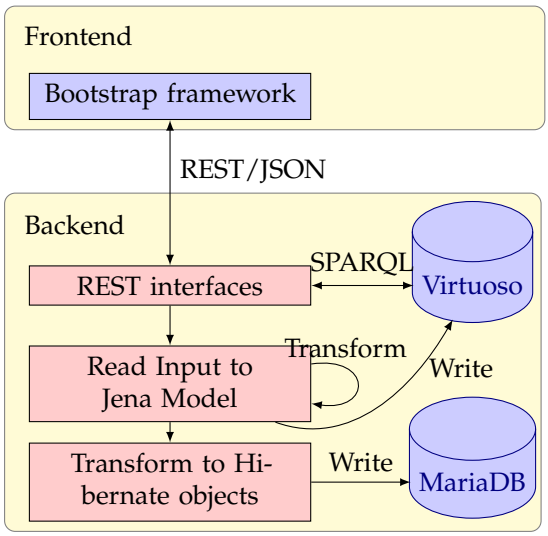


Figure 20.2: Visualization of front and back end to store the mappings in the Virtuoso and MariaDB

The front end of our repository provides an easy way to use some of the functionality of LinkLion (see Figure 20.3a). First, it allows users to upload new mappings. Users are asked to provide a source file in the N-Triples format.[8] Moreover, the framework used to generate the links as well as the algorithm used within this framework have to be provided (note that we consider humans to also be linking frameworks). The data given by the user is then checked for consistency and uploaded into the underlying triple store. The content of the triple store can be browsed directly from the web page (see Figure 20.3b). Especially, the front end includes search functionality and pagination which allow end users to search for mappings that link to or from a dataset of interest. The upload and browsing functionality will be presented during the demo.

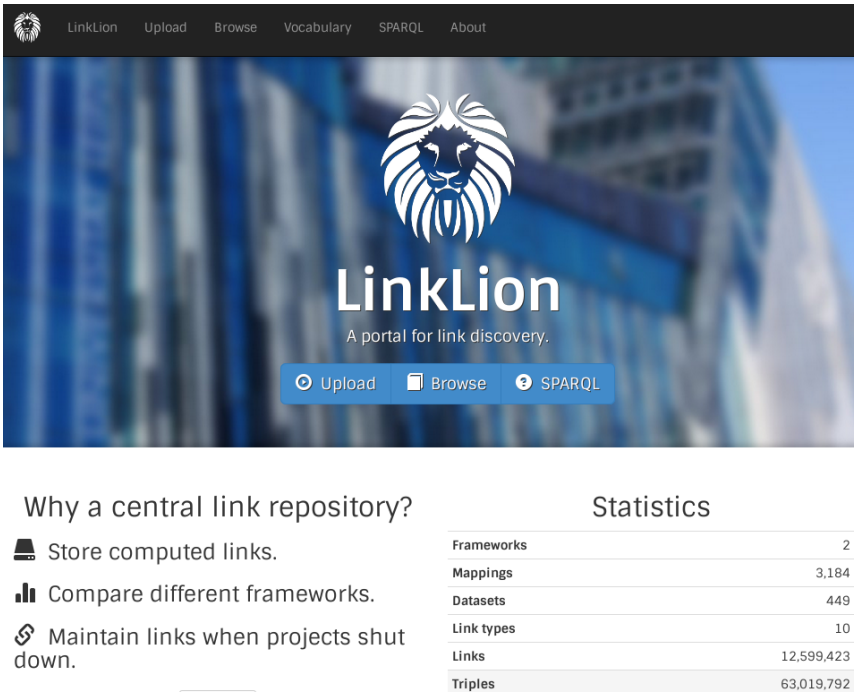
20.3 USE CASES

In this section, we present and motivate a selection of use cases that will be presented during the demo session.

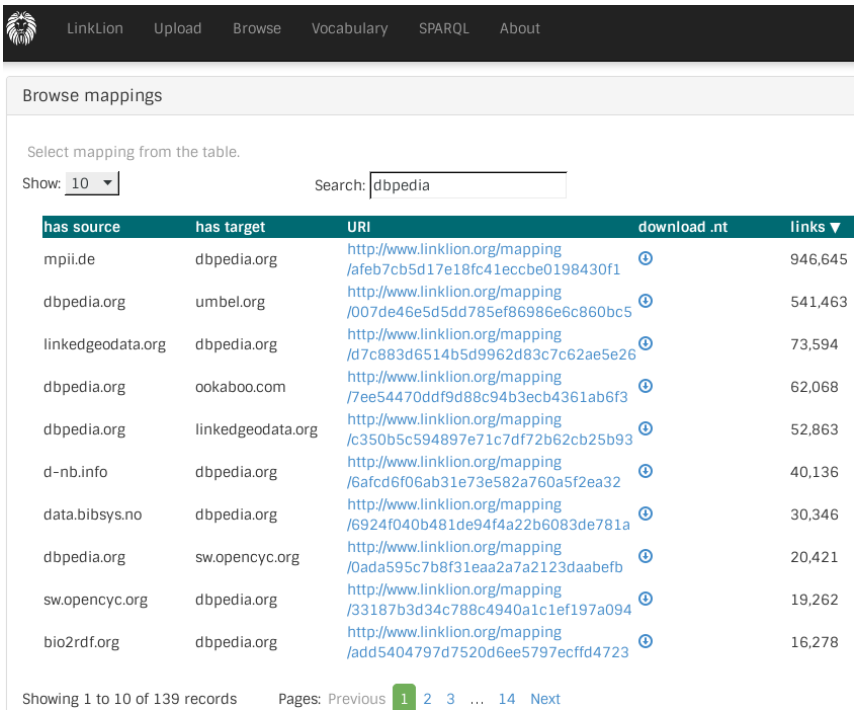
20.3.1 *Gather all links and mappings to a given resource*

Gathering and fusing all information on a resource of interest is of central importance to applications such as Question Answering systems, Linked Data Browsers and Quality Assessment tools. The LinkLion portal allows gathering all links pertaining to a particular resource (`dbpedia:Thailand` in our example) by means of the following SPARQL query. By using this information, novel repair-based algorithms for link discovery such as Colibri can find errors or inconsistencies in the data (Ngonga Ngomo et al., 2014).

8 http://www.w3.org/2001/sw/RDFCore/ntriples/



(a) LinkLion Homepage with statistics regarding the content.



(b) Mapping Browser. The search for DBpedia returns 139 mappings.

Figure 20.3: Front-end views

Listing 20.1: Query to find all links to Thailand

```
SELECT ?link WHERE { { ?link rdf:subject dbpedia:Thailand }
            UNION { ?link rdf:object dbpedia:Thailand } }
```

The portal also allows gather all mappings that contain links pertaining to a particular resource, e.g., `dbpedia:Thailand`, as shown below.

Listing 20.2: Gather all mapping pertaining to Thailand

```
SELECT DISTINCT ?mapping WHERE { ?link prov:wasDerivedFrom ?mapping .
                                { ?link rdf:subject dbpedia:Thailand }
                          UNION { ?link rdf:object dbpedia:Thailand } }
```

20.3.2 *Get support for a link*

Ensemble learning techniques have been shown to improve the results of manifold machine-learning applications such as Named Entity Recognition frameworks. Our repository facilitates the use of ensemble learning for combining the results of different link discovery tools. Especially, LinkLion allows us to retrieve (if any) the list of mappings that contain a given link, as well as the algorithms and the frameworks that generated it. In the following example, the support for `dbpedia:Thailand owl:sameAs <http://sws.geonames.org/1605651/>` is queried.

Listing 20.3: Query to find the support of a link

```
SELECT ?mapping ?algorithm ?framework WHERE {
        ?mapping prov:wasGeneratedBy ?algorithm .
        ?algorithm prov:wasAssociatedWith ?framework .
        ?link prov:wasDerivedFrom ?mapping ;
              rdf:predicate owl:sameAs .
        { ?link rdf:subject dbpedia:Thailand;
                rdf:object <http://sws.geonames.org/1605651/> }
        UNION { ?link rdf:object dbpedia:Thailand;
                      rdf:subject <http://sws.geonames.org/1605651/> }
                          }
```

20.3.3 *Link Composition*

With the growth of the Linked Data Web, it becomes ever more important to regard link discovery as a holistic process that goes beyond linking pair of knowledge bases. Algorithms based on composition can exploit sequences of links to enrich their mapping composition graphs (Hartung et al., 2013). Moreover, algorithms which link several knowledge bases at the same time (Ngonga Ngomo et al., 2014) can achieve higher accuracies. By using LinkLion, composition and concurrent linking algorithms are now enabled to gather the data they require without having to manage all the links by themselves. In the query below, all resources related to `dbpedia:Thailand` over two links are retrieved from the repository. By using such link paths, we can easily detect and correct inconsistent links in the portal or even learn specifications to compute links cross manifold knowledge bases.

Listing 20.4: Query to retrieve all resources related to Thailand over two links

```
SELECT DISTINCT ?resource WHERE { {
        ?link rdf:subject dbpedia:Thailand ; rdf:object ?x .
        ?link2 rdf:subject ?x ; rdf:object ?resource }
        UNION { ?link rdf:object dbpedia:Thailand ; rdf:subject ?x .
                ?link2 rdf:object ?x ; rdf:subject ?resource } }
```

Part V

APPLICATIONS

The presentation of LIMES and the corresponding implementation is now concluded. The goal of this chapter is to present a number of applications of the approaches included in LIMES. We begin by presenting two triple store benchmarks (see Chapter 21 and Chapter 22) that rely on LIMES algorithms or extensions thereof to compare queries gathered from query logs. We then show how LIMES was used to create the data layer for the question answering engine DEQA (see Chapter 23). The application of our approaches for open knowledge extraction is presented in Chapter 24. The remaining chapters present datasets whose generation process included LIMES for the link discovery step. We chose diverse datasets to underline the applicability of our framework.

21 THE DBPEDIA SPARQL BENCHMARK

PREAMBLE

Triple stores are the backbone of increasingly many Data Web applications. The performance of those stores is thus mission-critical for Web applications. In this chapter, we present a benchmark for triple stores that relies on LIMES to compute similar queries. The content of this chapter is taken from (Morsey et al., 2011, 2012). The author co-designed the process from query logs to benchmarks and implemented the query comparison. Moreover, the author co-wrote the papers.

21.1 INTRODUCTION

Triple stores, which use IRIs for entity identification and store information adhering to the RDF data model (Klyne and Carroll, 2004) are the backbone of increasingly many Data Web applications. The RDF data model resembles directed labeled graphs, in which each labeled edge (called *predicate*) connects a *subject* to an *object*. The intended semantics is that the *object* denotes the value of the *subject's* property *predicate*. With the W3C SPARQL standard (Prud'hommeaux and Seaborne, 2008) a vendor-independent query language for the RDF triple data model exists. SPARQL is based on powerful graph matching allowing to bind variables to fragments in the input RDF graph. In addition, operators akin to the relational joins, unions, left outer joins, selections and projections can be used to build more expressive queries (Schmidt et al., 2009). It is evident that the performance of triple stores offering a SPARQL query interface is mission critical for individual projects as well as for data integration on the Web in general. It is consequently of central importance during the implementation of any Data Web application to have a clear picture of the weaknesses and strengths of current triple store implementations.

Existing SPARQL benchmark efforts such as LUBM (Guo et al., 2005), BSBM (Bizer and Schultz, 2009) and SP^2 (Schmidt et al., 2009) resemble relational database benchmarks. Especially the data structures underlying these benchmarks are basically relational data structures, with relatively few and homogeneously structured classes. However, RDF knowledge bases are increasingly heterogeneous. Thus, they do not resemble relational structures and are not easily representable as such. Examples of such knowledge bases are curated bio-medical ontologies such as those contained in Bio2RDF (Belleau et al., 2008) as well as knowledge bases extracted from unstructured or semi-structured sources such as DBpedia (Auer et al., 2007) or LinkedGeoData (Auer et al., 2009). DBpedia (version 3.6) for example contains 289,016 classes of which 275 classes belong to the DBpedia ontology. Moreover, it contains 42,016 properties, of which 1335 are DBpedia-specific. Also, various datatypes and object references of different types are used in property values. Such knowledge bases can *not* be easily represented according to the relational data model and hence performance characteristics for loading, querying and up-

dating these knowledge bases might potentially be fundamentally different from knowledge bases resembling relational data structures.

In this article, we propose a generic SPARQL benchmark creation methodology. This methodology is based on a flexible data generation mimicking an input data source, query-log mining, clustering and SPARQL feature analysis. We apply the proposed methodology to datasets of various sizes derived from the DBpedia knowledge base. In contrast to previous benchmarks, we perform measurements on *real* queries that were issued by humans or Data Web applications against existing RDF data. We evaluate two different methods *data generation* approaches and show how a representative set of resources that preserves important dataset characteristics such as indegree and outdegree can be obtained by sampling across classes in the dataset. In order to obtain a representative set of *prototypical queries* reflecting the typical workload of a SPARQL endpoint, we perform a query analysis and clustering on queries that were sent to the official DBpedia SPARQL endpoint. From the highest-ranked query clusters (in terms of aggregated query frequency), we derive a set of 25 SPARQL query templates, which cover most commonly used SPARQL features and are used to generate the actual benchmark queries by parametrization. We call the benchmark resulting from this dataset and query generation methodology *DBPSB* (i.e. DBpedia SPARQL Benchmark). The benchmark methodology and results are also available online.[1] Although we apply this methodology to the DBpedia dataset and its SPARQL query log in this case, the same methodology can be used to obtain application-specific benchmarks for other knowledge bases and query workloads. Since the DBPSB can change with the data and queries in DBpedia, we envision to update it in yearly increments and publish results on the above website. In general, our methodology follows the four key requirements for domain specific benchmarks are postulated in the Benchmark Handbook (Gray, 1991), i.e., it is (1) relevant, thus testing typical operations within the specific domain, (2) portable, i.e. executable on different platforms, (3) scalable, e.g. it is possible to run the benchmark on both small and very large datasets, and (4) it is understandable.

We apply the DBPSB to assess the performance and scalability of the popular triple stores *Virtuoso* (Erling and Mikhailov, 2007), *Sesame* (Broekstra et al., 2002), *Jena-TDB* (Owens et al., 2008), and *BigOWLIM* (Bishop et al., 2011) and compare our results with those obtained with previous benchmarks. Our experiments reveal that the performance and scalability is by far less homogeneous than other benchmarks indicate. As we explain in more detail later, we believe this is due to the different nature of DBPSB compared to the previous approaches resembling relational databases benchmarks. For example, we observed query performance differences of several orders of magnitude much more often than with other RDF benchmarks when looking at the runtimes of individual queries. The main observation in our benchmark is that previously observed differences in performance between different triple stores amplify when they are confronted with actually asked SPARQL queries, i.e. there is now a wider gap in performance compared to essentially relational benchmarks.

The remainder of the paper is organized as follows. In Section 21.2, we describe the dataset generation process in detail. We show the process of query analysis and clustering in detail in Section 21.3. In Section 21.4, we present our approach to selecting SPARQL features and to query variability. The assessment of four triple

1 http://aksw.org/Projects/DBPSB

Dataset	Indegree w/ literals	Outdegree w/ literals	Indegree w/o literals	Outdegree w/o literals	No. of nodes	No. of triples
Full DBpedia	5.45	30.52	3.09	15.57	27,665,352	153,737,776
10% dataset (seed)	6.54	45.53	3.98	23.05	2,090,714	15,267,418
10% dataset (rand)	3.82	6.76	2.04	3.41	5,260,753	16,739,055
50% dataset (seed)	6.79	38.08	3.82	18.64	11,317,362	74,889,154
50% dataset (rand)	7.09	26.79	3.33	10.73	9,581,470	78,336,781

Table 21.1: Statistical analysis of DBPSB datasets

stores via the DBPSB is described in Section 21.5 and Section 21.6. The results of the experiment are discussed in Section 21.7. We present related work in Section 21.8 and conclude our paper in Section 21.9.

21.2 DATASET GENERATION

A crucial step in each benchmark is the generation of suitable datasets. Although we describe the dataset generation here with the example of DBpedia, the methodology we pursue is dataset-agnostic.

The data generation for DBPSB is guided by the following requirements:

- The DBPSB data should resemble the original data (i.e., DBpedia data in our case) as much as possible, in particular the large number of classes, properties, the heterogeneous property value spaces as well as the large taxonomic structures of the category system should be preserved.
- The data generation process should allow to generate knowledge bases of various sizes ranging from a few million to several hundred million or even billion triples.
- Basic network characteristics of different sizes of the network should be similar, in particular the in- and outdegree.
- The data generation process should be easily repeatable with new versions of the considered dataset.

The proposed dataset creation process starts with an input dataset. For the case of DBpedia, it consists of the datasets loaded into the official SPARQL endpoint.[2] Datasets of multiple size of the original data are created by duplicating all triples and changing their namespaces. This procedure can be applied for any scale factors. While simple, this procedure is efficient to execute and fulfills the above requirements.

For generating smaller datasets, we investigated two different methods. The first method (called "rand") consists of selecting an appropriate fraction of all triples of the original dataset randomly. If RDF graphs are considered as small world graphs, removing edges in such graphs should preserve the properties of the original graph. The second method (called "seed") is based on the assumption that a representative set of resources can be obtained by sampling across classes in the

2 Endpoint: http://dbpedia.org/sparql, Loaded datasets: http://wiki.dbpedia.org/DatasetsLoaded

dataset. Let x be the desired scale factor in percent, e.g. $x = 10$. The method first selects x% of the classes in the dataset. For each selected class, 10% of its instances are retrieved and added to a queue. For each element of the queue, its concise bound description (CBD)[3] is retrieved. This can lead to new resources, which are appended at the end of the queue. This process is iterated until the target dataset size, measured in number of triples, is reached.

Since the selection of the appropriate method for generating small datasets is an important issue, we performed a statistical analysis on the generated datasets for DBpedia. The statistical parameters used to judge the datasets are the average indegree, the average outdegree, and the number of nodes, i.e., number of distinct IRIs in the graph. We calculated both the in- and the outdegree for datasets once with literals ignored, and another time with literals taken into consideration, as it gives more insight on the degree of similarity between the dataset of interest and the full DBpedia dataset. The statistics of those datasets are given in Table 21.1. According to this analysis, the seed method fits our purpose of maintaining basic network characteristics better, as the average in- and outdegree of nodes are closer to the original dataset. For this reason, we selected this method for generating the DBPSB.

21.3 QUERY ANALYSIS AND CLUSTERING

The goal of the query analysis and clustering is to detect prototypical queries that were sent to the official DBpedia SPARQL endpoint based on a query-similarity graph. Note that two types of similarity measures can been used on queries, i. e. string similarities and graph similarities. Yet, since graph similarities are very time-consuming and do not bear the specific mathematical characteristics necessary to compute similarity scores efficiently, we picked string similarities for our experiments. In the query analysis and clustering step, we follow a four-step approach. First, we select queries that were executed frequently on the input data source. Second, we strip common syntactic constructs (e.g., namespace prefix definitions) from these query strings in order to increase the conciseness of the query strings. Then, we compute a query similarity graph from the stripped queries. Finally, we use a soft graph clustering algorithm for computing clusters on this graph. These clusters are subsequently used to devise the query generation patterns used in the benchmark. In the following, we describe each of the four steps in more detail.

QUERY SELECTION For the DBPSB, we use the DBpedia SPARQL query-log which contains all queries posed to the official DBpedia SPARQL endpoint for a three-month period in 2010[4]. For the generation of the current benchmark, we used the log for the period from April to July 2010. Overall, 31.5 million queries were posed to the endpoint within this period. In order to obtain a small number of distinctive queries for benchmarking triple stores, we reduce those queries in the following two ways:

- *Query variations.* Often, the same or slight variations of the same query are posed to the endpoint frequently. A particular cause of this is the renaming of query variables. We solve this issue by renaming all query variables in a

3 http://www.w3.org/Submission/CBD/. Retrieved July 28th, 2015.

4 The DBpedia SPARQL endpoint is available at: http://dbpedia.org/sparql/ and the query log excerpt at: ftp://download.openlinksw.com/support/dbpedia/.

consecutive sequence as they appear in the query, i.e., *var0*, *var1*, *var2*, and so on. As a result, distinguishing query constructs such as `REGEX` or `DISTINCT` are a higher influence on the clustering.

- *Query frequency.* We discard queries with a low frequency (below 10) because they do not contribute much to the overall query performance.

The application of both methods to the query log dataset at hand reduced the number of queries from 31.5 million to just 35,965. This reduction allows our benchmark to capture the essence of the queries posed to DBpedia within the timespan covered by the query log and reduces the runtime of the subsequent steps substantially.

STRING STRIPPING Every SPARQL query contains substrings that segment it into different clauses. Although these strings are essential during the evaluation of the query, they are a major source of noise when computing query similarity, as they boost the similarity score without the query patterns being similar per se. Therefore, we remove all SPARQL syntax keywords such as `PREFIX`, `SELECT`, `FROM` and `WHERE`. In addition, common prefixes (such as `http://www.w3.org/2000/01/rdf-schema#` for RDF-Schema) are removed as they appear in most queries.

SIMILARITY COMPUTATION The goal of the third step is to compute the similarity of the stripped queries. Computing the Cartesian product of the queries would lead to a quadratic runtime, i.e., almost 1.3 billion similarity computations. To reduce the runtime of the benchmark compilation, we use the LIMES framework (Ngonga Ngomo and Auer, 2011).[5] The LIMES approach makes use of the interchangeability of similarities and distances. It presupposes a metric space in which the queries are expressed as single points. Instead of aiming to find all pairs of queries such that $\mathrm{sim}(q, p) \geqslant \theta$, LIMES aims to find all pairs of queries such that $d(q, p) \leqslant \tau$, where sim is a similarity measure and d is the corresponding metric. To achieve this goal, when given a set of n queries, it first computes $\sqrt{n}$ so-called *exemplars*, which are prototypical points in the affine space that subdivide it into regions of high heterogeneity. Then, each query is mapped to the exemplar it is least distant to. The characteristics of metrics spaces (especially the triangle inequality) ensures that the distances from each query q to any other query p obeys the following inequality

$$d(q, e) - d(e, p) \leqslant d(q, p) \leqslant d(q, e) + d(e, p), \tag{21.1}$$

where e is an exemplar and d is a metric. Consequently,

$$d(q, e) - d(e, p) > \tau \Rightarrow d(q, p) > \tau. \tag{21.2}$$

Given that $d(q, e)$ is constant, q must only be compared to the elements of the list of queries mapped to e that fulfill the inequality above. By these means, the number of similarity computation can be reduced significantly. In this particular use case, we cut down the number of computations to only 16.6% of the Cartesian product without any loss in recall. For the current version of the benchmark, we used the *Levenshtein* string similarity measure and a threshold of 0.9.

5 Available online at: `http://limes.sf.net`

CLUSTERING The final step of our approach is to apply graph clustering to the query similarity graph computed above. The goal of this step is to discover very similar groups queries out of which prototypical queries can be generated. As a given query can obey the patterns of more than one prototypical query, we opt for using the soft clustering approach implemented by the BorderFlow algorithm.[6]

BorderFlow (Ngonga Ngomo and Schumacher, 2009) implements a seed-based approach to graph clustering. The default setting for the seeds consists of taking all nodes in the input graph as seeds. For each seed v, the algorithm begins with an initial cluster X containing only v. Then, it expands X iteratively by adding nodes from the direct neighborhood of X to X until X is node-maximal with respect to a function called the border flow ratio. The same procedure is repeated over all seeds. As different seeds can lead to the same cluster, identical clusters (i.e., clusters containing exactly the same nodes) that resulted from different seeds are subsequently collapsed to one cluster. The set of collapsed clusters and the mapping between each cluster and its seeds are returned as result. Applying BorderFlow to the input queries led to 12272 clusters, of which 24% contained only one node, hinting towards a long-tail distribution of query types. To generate the patterns used in the benchmark, we only considered clusters of size 5 and above.

21.4 SPARQL FEATURE SELECTION AND QUERY VARIABILITY

After the completion of the detection of similar queries and their clustering, our aim is now to select a number of frequently executed queries that cover most SPARQL features and allow us to assess the performance of queries with single as well as combinations of features. The SPARQL features we consider are:

- the overall number of triple patterns contained in the query (|GP|),
- the graph pattern constructors `UNION` (*UON*), `OPTIONAL` (*OPT*),
- the solution sequences and modifiers `DISTINCT` (*DST*),
- as well as the filter conditions and operators `FILTER` (*FLT*), `LANG` (*LNG*), `REGEX` (*REG*) and `STR` (*STR*).

We pick different numbers of triple patterns in order to include the efficiency of JOIN operations in triple stores. The other features were selected because they frequently occurred in the query log. We rank the clusters by the sum of the frequency of all queries they contain. Thereafter, we select 25 queries as follows: For each of the features, we choose the highest ranked cluster containing queries having this feature. From that particular cluster we select the query with the highest frequency.

In order to convert the selected queries into query templates, we manually select a part of the query to be varied. This is usually an IRI, a literal or a filter condition. In Figure 21.1 those varying parts are indicated by `%%v%%` or in the case of multiple varying parts `%%vn%%`. We exemplify our approach to replacing varying parts of queries by using Query 9, which results in the query shown in Figure 21.1. This query selects a specific settlement along with the airport belonging to that settlement as indicated in Figure 21.1. The variability of this query template was determined by getting a list of all settlements using the query shown in Figure 21.2. By selecting suitable placeholders, we ensured that the variability is sufficiently high

6 An implementation of the algorithm can be found at `http://borderflow.sf.net`

($\geqslant$ 1000 per query template). Note that the triple store used for computing the variability was different from the triple store that we later benchmarked in order to avoid potential caching effects.

```
SELECT * WHERE {
  { ?v2  a           dbp-owl:Settlement ;
         rdfs:label  %%v%% .
    ?v6  a           dbp-owl:Airport . }
  { ?v6  dbp-owl:city  ?v2 . }
  UNION
  { ?v6  dbp-owl:location  ?v2 . }
  { ?v6  dbp-prop:iata     ?v5 . }
  UNION
  { ?v6  dbp-owl:iataLocationIdentifier  ?v5 . }
  OPTIONAL { ?v6  foaf:homepage  ?v7 . }
  OPTIONAL { ?v6  dbp-prop:nativename  ?v8 . }
}
```

Figure 21.1: Sample query with placeholder

```
SELECT DISTINCT ?v WHERE {
  { ?v2  a           dbp-owl:Settlement ;
         rdfs:label  ?v .
    ?v6  a           dbp-owl:Airport . }
  { ?v6  dbp-owl:city  ?v2 . }
  UNION
        { ?v6  dbp-owl:location  ?v2 . }
  { ?v6  dbp-prop:iata     ?v5 . }
  UNION
        { ?v6  dbp-owl:iataLocationIdentifier ?v5 . }
  OPTIONAL { ?v6  foaf:homepage  ?v7 . }
  OPTIONAL { ?v6  dbp-prop:nativename  ?v8 . }
} LIMIT 1000
```

Figure 21.2: Sample auxiliary query returning potential values a placeholder can assume

For the benchmarking we then used the list of thus retrieved concrete values to replace the %%v%% placeholders within the query template. This method ensures, that (a) the actually executed queries during the benchmarking differ, but (b) always return results. This change imposed on the original query avoids the effect of simple caching.

21.5 EXPERIMENTAL SETUP

This section presents the setup we used when applying the DBPSB on four triple stores commonly used in Data Web applications. We first describe the triple stores and their configuration, followed by our experimental strategy and finally the obtained results. All experiments were conducted on a typical server machine with an AMD Opteron 6 Core CPU with 2.8 GHz, 32 GB RAM, 3 TB RAID-5 HDD running Linux Kernel 2.6.35-23-server and Java 1.6 installed. The benchmark program and the triple store were run on the same machine to avoid network latency.

TRIPLE STORES SETUP We carried out our experiments by using the triple stores Virtuoso, Sesame, Jena-TDB and BigOWLIM. The configuration and the version of each triple store were as follows:

1. **Virtuoso** Open-Source Edition version 6.1.2: We set the following memory-related parameters: NumberOfBuffers = 1048576, MaxDirtyBuffers = 786432.
2. **Sesame** Version 2.3.2 with Tomcat 6.0 as HTTP interface: We used the native storage layout and set the spoc, posc, opsc indices in the native storage configuration. We set the Java heap size to 8GB.
3. **Jena-TDB** Version 0.8.7 with Joseki 3.4.3 as HTTP interface: We configured the TDB optimizer to use statistics. This mode is most commonly employed for the TDB optimizer, whereas the other modes are mainly used for investigating the optimizer strategy. We also set the Java heap size to 8GB.
4. **BigOWLIM** Version 3.4, with Tomcat 6.0 as HTTP interface: We set the entity index size to 45,000,000 and enabled the predicate list. The rule set was empty. We set the Java heap size to 8GB.

In summary, we configured all triple stores to use 8GB of memory and used default values otherwise. This strategy aims on the one hand at benchmarking each triple store in a real context, as in real environment a triple store cannot dispose of the whole memory up. On the other hand it ensures that the whole dataset cannot fit into memory, in order to avoid caching.

BENCHMARK EXECUTION Once the triple stores loaded the DBpedia datasets with different scale factors, i.e. 10%, 50%, 100%, and 200%, the benchmark execution phase began. It comprised the following stages:

1. **System Restart**: Before running the experiment, the triple store and its associated programs were restarted in order to clear memory caches.
2. **Warm-up Phase**: In order to measure the performance of a triple store under normal operational conditions, a warm-up phase was used. In the warm-up phase, query mixes were posed to the triple store. The queries posed during the warm-up phase were disjoint with the queries posed in the hot-run phase. For DBPSB, we used a warm-up period of 20 minutes.
3. **Hot-run Phase**: During this phase, the benchmark query mixes were sent to the tested store. We kept track of the average execution time of each query as well as the number of query mixes per hour (QMpH). The duration of the hot-run phase in DBPSB was 60 minutes.

Since some benchmark queries did not respond within reasonable time, we specified a 180 second timeout after which a query was aborted and the 180 second maximum query time was used as the runtime for the given query even though no results were returned. The benchmarking code along with the DBPSB queries is freely available.[7]

7 https://akswbenchmark.svn.sourceforge.net/svnroot/akswbenchmark/

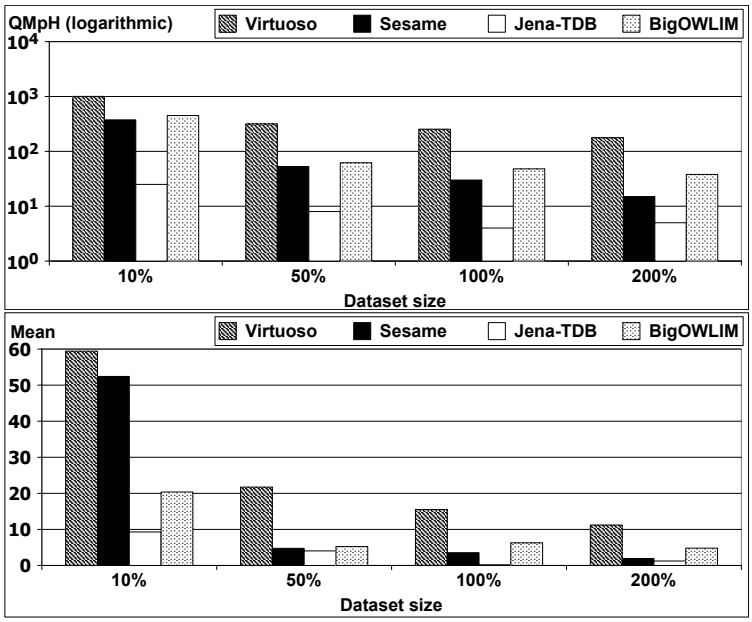


Figure 21.3: QMpH for all triple stores (top). Geometric mean of QpS (bottom).

21.6 RESULTS

We evaluated the performance of the triple stores with respect to two main metrics: their overall performance on the benchmark and their query-based performance.

The overall performance of any triple store was measured by computing its query mixes per hour (QMpH) as shown in Figure 21.3. Please note that we used a logarithmic scale in this figure due to the high performance differences we observed. In general, Virtuoso was clearly the fastest triple store, followed by BigOWLIM, Sesame and Jena-TDB. The highest observed ratio in QMpH between the fastest and slowest triple store was 63.5 and it reached more than 10 000 for single queries. The scalability of stores did not vary as much as the overall performance. There was on average a linear decline in query performance with increasing dataset size. Details will be discussed in Section 21.7.

We tested the queries that each triple store failed to executed withing the 180s timeout and noticed that even much larger timeouts would not have been sufficient most of those queries. We did not exclude the queries completely from the overall assessment, since this would have affected a large number of the queries and adversely penalized stores, which complete queries within the time frame. We penalized failure queries with 180s, similar to what was done in the SP2-Benchmark (Schmidt et al., 2009). Virtuoso was the only store, which completed all queries in time. For Sesame and OWLIM only rarely a few particular queries timed out. Jena-TDB had always severe problems with queries 7, 10 and 20 as well as 3, 9, 12 for the larger two datasets.

The metric used for query-based performance evaluation is Queries per Second (QpS). QpS is computed by summing up the runtime of each query in each iteration, dividing it by the QMpH value and scaling it to seconds. The QpS results for all triple stores and for the 10%, 50%, 100%, and 200% datasets are depicted in Figure 21.5.

The outliers, i.e. queries with very low QpS, will significantly affect the mean value of QpS for each store. So, we additionally calculated the geometric mean of all the QpS timings of queries for each store. The geometric mean for all triple stores is also depicted in Figure 21.3. By reducing the effect of outliers, we obtained additional information from this figure as we will describe in the subsequent section.

21.7 DISCUSSION

This section consists of three parts: First, we compare the general performance of the systems under test. Then we look individual queries and the SPARQL features used within those queries in more detail to observe particular strengths and weaknesses of stores. Thereafter, we compare our results with those obtained with previous benchmarks and elucidate some of the main differences between them.

GENERAL PERFORMANCE Figure 21.3 depicts the benchmark results for query mixes per hour for the four systems and dataset sizes. Virtuoso leads the field with a substantial head start of double the performance for the 10% dataset (and even quadruple for other dataset sizes) compared to the second best system (BigOWLIM). While Sesame is able to keep up with BigOWLIM for the smaller two datasets it considerably looses ground for the larger datasets. Jena-TDB can in general not deliver competitive performance with being by a factor 30-50 slower than the fastest system.

If we look at the geometric mean of all QpS results in Figure 21.3, we observe similar insights. The spreading effect is weakened, since the geometric mean reduces the effect of outliers. Still Virtuoso is the fastest system, although Sesame manages to get pretty close for the 10% dataset. This shows that most, but not all, queries are fast in Sesame for low dataset sizes. For the larger datasets, BigOWLIM is the second best system and shows promising scalability, but it is still by a factor of two slower than Virtuoso.

SCALABILITY, INDIVIDUAL QUERIES AND SPARQL FEATURES Our first observation with respect to individual performance of the triple stores is that Virtuoso demonstrates a good scaling factor on the DBPSB. When dataset size changes by factor 5 (from 10% to 50%), the performance of the triple store only degrades by factor 3.12. Further dataset increases (i.e. the doubling to the 100% and 200% datasets) result in only relatively small performance decreases by 20% and respectively 30%.

Virtuoso outperforms Sesame for all datasets. In addition, Sesame does not scale as well as Virtuoso for small dataset sizes, as its performance degrades sevenfold when the dataset size changes from 10% to 50%. However, when the dataset size doubles from the 50% to the 100% dataset and from 100% to 200% the performance degrades by just half.

The performance of Jena-TDB is the lowest of all triple stores and for all dataset sizes. The performance degradation factor of Jena-TDB is not as pronounced as that of Sesame and almost equal to that of Virtuoso when changing from the 10% to the 50% dataset. However, the performance of Jena-TDB only degrades by a factor of 2 for the transition between the 50% and 100% dataset, and reaches 0.8 between the 100% and 200% dataset, leading to a slight increase of its QMpH.

BigOWLIM is the second fastest triple store for all dataset sizes, after Virtuoso. BigOWLIM degrades with a factor of 7.2 in transition from 10% to 50% datasets, but it decreases dramatically to 1.29 with dataset size 100%, and eventually reaches 1.26 with dataset size 200%.

Due to the high diversity in the performance of different SPARQL queries, we also computed the geometric mean of the QpS values of all queries as described in the previous section and illustrated in Figure 21.3. By using the geometric mean, the resulting values are less prone to be dominated by a few outliers (slow queries) compared to standard QMpH values. This allows for some interesting observations in DBPSB by comparing Figure 21.3 and Figure 21.3. For instance, it is evident that Virtuoso has the best QpS values for all dataset sizes.

With respect to Virtuoso, query 10 performs quite poorly. This query involves the features `FILTER`, `DISTINCT`, as well as `OPTIONAL`. Also, the well performing query 1 involves the `DISTINCT` feature. Query 3 involves a `OPTIONAL` resulting in worse performance. Query 2 involving a `FILTER` condition results in the worst performance of all of them. This indicates that using complex `FILTER` in conjunction with additional `OPTIONAL`, and `DISTINCT` adversely affects the overall runtime of the query.

Regarding Sesame, queries 4 and 18 are the slowest queries. Query 4 includes `UNION` along with several free variables, which indicates that using `UNION` with several free variables causes problems for Sesame. Query 18 involves the features `UNION`, `FILTER`, `STR` and `LANG`. Query 15 involves the features `UNION`, `FILTER`, and `LANG`, and its performance is also pretty slow, which leads to the conclusion that introducing this combination of features is difficult for Sesame. Adding the `STR` feature to that feature combination affects the performance dramatically and prevents the query from being successfully executed.

For Jena-TDB, there are several queries that timeout with large dataset sizes, but queries 10 and 20 always timeout. The problem with query 10 is already discussed with Virtuoso. Query 20 contains `FILTER`, `OPTIONAL`, `UNION`, and `LANG`. Query 2 contains `FILTER` only, query 3 contains `OPTIONAL`, and query 4 contains `UNION` only. All of those queries run smoothly with Jena-TDB, which indicates that using the `LANG` feature, along with those features affects the runtime dramatically.

For BigOWLIM, queries 10, and 15 are slow queries. Query 10 was already problematic for Virtuoso, as was query 15 for Sesame.

Query 24 is slow on Virtuoso, Sesame, and BigOWLIM, whereas it is faster on Jena-TDB. This is due to the fact that most of the time this query returns many results. Virtuoso, and BigOWLIM return a bulk of results at once, which takes long time. Jena-TDB just returns the first result as a starting point, and iteratively returns the remaining results via a buffer.

It is interesting to note that BigOWLIM shows in general good performance, but almost never manages to outperform any of the other stores. Queries 11, 13, 19, 21 and 25 were performed with relatively similar results across triple stores thus indicating that the features of these queries (i.e. UON, REG, FLT) are already relatively well supported. With queries 3, 4, 7, 9, 12, 18, 20 we observed dramatic differences between the different implementations with factors between slowest and fastest store being higher than 1000. It seems that a reason for this could be the poor support for OPT (in queries 3, 7, 9, 20) as well as certain filter conditions such as LNG in some implementations, which demonstrates the need for further optimizations.

COMPARISON WITH PREVIOUS BENCHMARKS In order to visualize the performance improvement or degradation of a certain triple store compared to its competitors, we calculated the relative performance for each store compared to the average and depicted it for each dataset size in Figure 21.4. We also performed this calculation for BSBM version 2 and version 3. Overall, the benchmarking results with DBPSB were less homogeneous than the results of previous benchmarks. While with other benchmarks the ratio between fastest and slowest query rarely exceeds a factor of 50, the factor for the DBPSB queries (derived from real DBpedia SPARQL endpoint queries) reaches more than 1 000 in some cases.

As with the other benchmarks, Virtuoso was also fastest in our measurements. However, the performance difference is even higher than reported previously: Virtuoso reaches a factor of 3 in our benchmark compared to 1.8 in BSBM V3. BSBM V2 and our benchmark both show that Sesame is more suited to smaller datasets and does not scale as well as other stores. Jena-TDB is the slowest store in BSBM V3 and DBPSB, but in our case they fall much further behind to the point that Jena-TDB can hardly be used for some of the queries, which are asked to DBpedia. The main observation in our benchmark is that previously observed differences in performance between different triple stores amplify when they are confronted with actually asked SPARQL queries, i.e. there is now a wider gap in performance compared to essentially relational benchmarks.

21.8 RELATED WORK

	LUBM	SP²Bench	BSBM V2	BSBM V3	DBPSB
RDF stores tested	DLDB-OWL, Sesame, OWL-JessKB	ARQ, Redland, SDB, Sesame, Virtuoso	Virtuoso, Sesame, Jena-TDB, Jena-SDB	Virtuoso, 4store, BigData Jena-TDB BigOwlim	Virtuoso, Jena-TDB, BigOWLIM Sesame
Test data	Synthetic	Synthetic	Synthetic	Synthetic	Real
Test queries	Synthetic	Synthetic	Synthetic	Synthetic	Real
Size of tested datasets	0.1M, 0.6M, 1.3M, 2.8M, 6.9M	10k, 50k, 250k, 1M,	1M, 25M, 100M, 5M, 25M	100M, 200M	14M, 75M, 150M, 300M
Dist. queries	14	12	12	12	25
Multi-client	–	–	x	x	–
Use case	Universities	DBLP	E-commerce	E-commerce	DBpedia
Classes	43	8	8	8	239 (internal) +300K(YAGO)
Properties	32	22	51	51	1200

Table 21.2: Comparison of different RDF benchmarks

Several RDF benchmarks were previously developed. The *Lehigh University Benchmark* (LUBM) (Guo et al., 2005) was one of the first RDF benchmarks. LUBM uses an artificial data generator, which generates synthetic data for universities, their departments, their professors, employees, courses and publications. This small number of classes limits the variability of data and makes LUMB inherent structure more repetitive. Moreover, the SPARQL queries used for benchmarking in LUBM are all plain queries, i.e. they contain only triple patterns with no other SPARQL features (e.g. `FILTER`, or `REGEX`). LUBM performs each query 10 consecu-

tive times, and then it calculates the average response time of that query. Executing the same query several times without introducing any variation enables query caching, which affects the overall average query times.

SP^{2}Bench (Schmidt et al., 2009) is another more recent benchmark for RDF stores. Its RDF data is based on the Digital Bibliography & Library Project (DBLP) and includes information about publications and their authors. It uses the SP^{2}Bench Generator to generate its synthetic test data, which is in its schema heterogeneity even more limited than LUMB. The main advantage of SP^{2}Bench over LUBM is that its test queries include a variety of SPARQL features (such as `FILTER`, and `OPTIONAL`). The main difference between the DBpedia benchmark and SP^{2}Bench is that both test data and queries are synthetic in SP^{2}Bench. In addition, SP^{2}Bench only published results for up to 25M triples, which is relatively small with regard to datasets such as DBpedia and LinkedGeoData.

Another benchmark described in (Owens et al., 2008) compares the performance of BigOWLIM and AllegroGraph. The size of its underlying synthetic dataset is 235 million triples, which is sufficiently large. The benchmark measures the performance of a variety of SPARQL constructs for both stores when running in single and in multi-threaded modes. It also measures the performance of adding data, both using bulk-adding and partitioned-adding. The downside of that benchmark is that it compares the performance of only two triple stores. Also the performance of each triple store is not assessed for different dataset sizes, which prevents scalability comparisons.

The Berlin SPARQL Benchmark (BSBM) (Bizer and Schultz, 2009) is a benchmark for RDF stores, which is applied to various triple stores, such as Sesame, Virtuoso, and Jena-TDB. It is based on an e-commerce use case in which a set of products is provided by a set of vendors and consumers post reviews regarding those products. It tests various SPARQL features on those triple stores. It tries to mimic a real user operation, i.e. it orders the query in a manner that resembles a real sequence of operations performed by a human user. This is an effective testing strategy. However, BSBM data and queries are artificial and the data schema is very homogeneous and resembles a relational database. This is reasonable for comparing the performance of triple stores with RDBMS, but does not give many insights regarding the specifics of RDF data management.

A comparison between benchmarks is shown in Table 21.2. In addition to general purpose RDF benchmarks it is reasonable to develop benchmarks for specific RDF data management aspects. One particular important feature in practical RDF triple store usage scenarios (as was also confirmed by DBPSB) is full-text search on RDF literals. In (Minack et al., 2009) the LUBM benchmark is extended with synthetic scalable fulltext data and corresponding queries for fulltext-related query performance evaluation. RDF stores are benchmarked for basic fulltext queries (classic IR queries) as well as hybrid queries (structured and fulltext queries).

21.9 CONCLUSIONS AND FUTURE WORK

We proposed the DBPSB benchmark for evaluating the performance of triple stores based on non-artificial data and queries. Our solution was implemented for the DBpedia dataset and tested with 4 different triple stores, namely Virtuoso, Sesame, Jena-TDB, and BigOWLIM. The main advantage of our benchmark over previous work is that it uses real RDF data with typical graph characteristics including a

large and heterogeneous schema part. Furthermore, by basing the benchmark on queries asked to DBpedia, we intend to spur innovation in triple store performance optimisation towards scenarios, which are actually important for end users and applications. We applied query analysis and clustering techniques to obtain a diverse set of queries corresponding to feature combinations of SPARQL queries. Query variability was introduced to render simple caching techniques of triple stores ineffective.

The benchmarking results we obtained reveal that real-world usage scenarios can have substantially different characteristics than the scenarios assumed by prior RDF benchmarks. Our results are more diverse and indicate less homogeneity than what is suggested by other benchmarks. The creativity and inaptness of real users while constructing SPARQL queries is reflected by DBPSB and unveils for a certain triple store and dataset size the most costly SPARQL feature combinations.

Several improvements can be envisioned in future work to cover a wider spectrum of features in DBPSB:

- Coverage of more SPARQL 1.1 features, e.g. reasoning and subqueries.
- Inclusion of further triple stores and continuous usage of the most recent DBpedia query logs.
- Testing of SPARQL update performance via DBpedia Live, which is modified several thousand times each day. In particular, an analysis of the dependency of query performance on the dataset update rate could be performed.

In addition, we will further investigate the data generation process, in particular based on recent work such as (Duan et al., 2011).

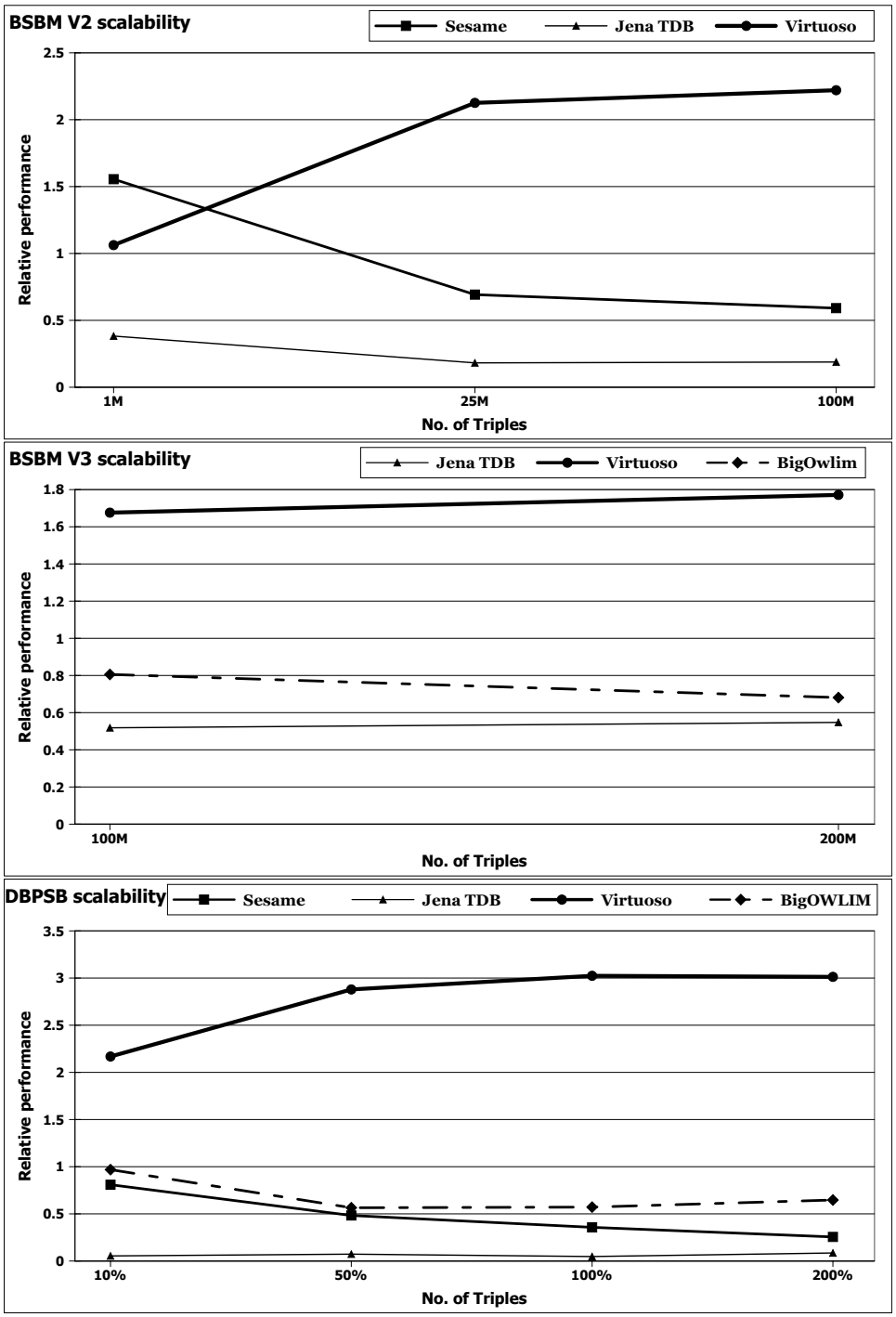


Figure 21.4: Comparison of triple store scalability between BSBM V2, BSBM V3 and DBPSB (from top to bottom)

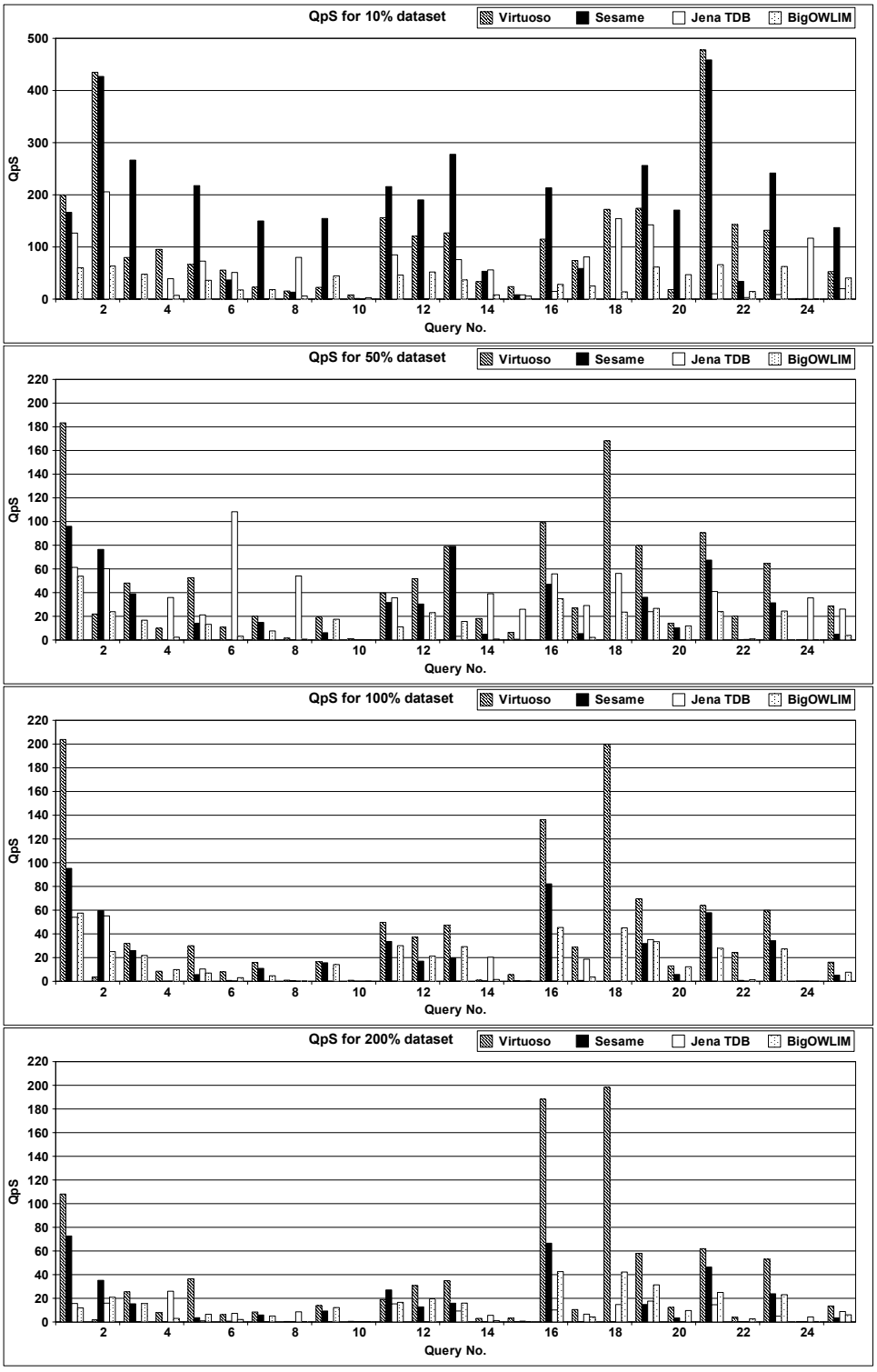


Figure 21.5: Queries per Second (QpS) for all triple stores for 10%, 50%, 100%, and 200% datasets (from top to bottom)

22 FEATURE-BASED GENERATION OF SPARQL BENCHMARKS

PREAMBLE

While the LIMES algorithm was used for similarity computation in Chapter 21, another application of the LIMES algorithm described in Chapter 3 is the portioning of Euclidean spaces for the sake of approximating query distributions from query logs. This chapter presents such an application and is taken from (Saleem et al., 2015). The author designed and implemented the core algorithm, co-designed the rest of the approach, co-wrote the paper and supervised the whole of the work.

22.1 INTRODUCTION

Benchmarking is indispensable when aiming to assess technologies with respect to their suitability for given tasks. While several benchmarks and benchmark generation frameworks have been developed to evaluate triple stores, they mostly provide a one-fits-all solution to the benchmarking problem. This approach to benchmarking is however unsuitable to evaluate the performance of a triple store for a given application with particular requirements. We address this drawback by presenting FEASIBLE, an automatic approach for the generation of benchmarks out of the query history of applications, i.e., query logs. The generation is achieved by selecting prototypical queries of a user-defined size from the input set of queries. We evaluate our approach on two query logs and show that the benchmarks it generates are accurate approximations of the input query logs. Moreover, we compare four different triple stores with benchmarks generated using our approach and show that they behave differently based on the data they contain and the types of queries posed. Our results suggest that FEASIBLE generates better sample queries than the state of the art. In addition, the better query selection and the larger set of query types used lead to triple store rankings which partly differ from the rankings generated by previous works.

22.2 INTRODUCTION

Triple stores are the data backbone of many Linked Data applications (Morsey et al., 2011, 2012). The performance of triple stores is hence of central importance for Linked-Data-based software ranging from real-time applications (Kamdar et al., 2014; Saleem et al., 2014) to on-the-fly data integration frameworks (Acosta et al., 2011; Saleem et al., 2013; Schwarte et al., 2011). Several benchmarks (e.g., (Aluç et al., 2014; Bizer and Schultz, 2009; Guo et al., 2005; Morsey et al., 2011; Schmidt et al., 2011, 2009)) for assessing the performance of the triple stores have been proposed. However, many of them (e.g., (Aluç et al., 2014; Bizer and Schultz, 2009; Guo et al., 2005; Schmidt et al., 2009)) rely on synthetic data or on synthetic queries. The main advantage of such synthetic benchmarks is that they commonly rely on data generators that can produce benchmarks of different data sizes and thus allow to test the scalability of triple stores. However, they often fail to reflect reality.

In particular, previous works (Duan et al., 2011) point out that artificial benchmarks are typically highly structured while real Linked Data sources are most commonly weakly structured. Moreover, synthetic queries most commonly fail to reflect the characteristics of the real queries sent to applications (Arias et al., 2011). Thus, synthetic benchmark results are rarely sufficient to detect the most suitable triple store for a particular real application. The DBpedia SPARQL Benchmark (DBPSB) (Morsey et al., 2011) addresses a portion of these drawbacks by evaluating the performance of triple stores based on real DBpedia query logs. The main drawback of this benchmark is however that it does not consider important data-driven and structural query features (e.g., number of join vertices, triple patterns selectivities or query execution times etc.) which greatly affect the performance of triple stores (Aluç et al., 2014; Görlitz et al., 2012) during the query selection process. Furthermore, it only considers `SELECT` queries. The other three basic SPARQL query forms, i.e., `ASK`, `CONSTRUCT`, and `DESCRIBE` are not included.

In this paper we present FEASIBLE, a benchmark generation framework able to generate benchmarks from a set of queries (in particular from query logs). Our approach aims to generate customized benchmarks for given use cases or needs of an application. To this end, FEASIBLE assumes that it is given a set of queries well as the number of queries (e.g., 25) to be included into the benchmark as input. Then, our approach computes a sample of the selected subset that reflects the distribution of the queries in the input set of queries. The resulting queries can then be fed to a benchmark execution framework to benchmark triple stores. The contributions of this work are as follows:

1. We present the first structure and data-driven feature-based benchmark generation approach from real queries. By comparing FEASIBLE with DBPSB, we show that considering data-driven and structural query features leads to benchmarks that are better approximations of the input set of queries.

2. We present a novel sampling approach for queries based based on exemplars (Ngonga Ngomo and Auer, 2011) and medoids.

3. Beside SPARQL `SELECT`, we conduct the first evaluation of 4 triple stores w.r.t. to their performance on `ASK`, `DESCRIBE` and `CONSTRUCT` queries separately.

4. We show that the performance of triple stores varies greatly across the four basic forms of SPARQL query. Moreover, we show that the features used by FEASIBLE allow for a more fine-grained analysis of our benchmarking results.

The rest of this paper is structured as follows: We begin by providing an overview of the key SPARQL query features that need to be considered while designing SPARQL benchmarks. Then, we compare existing benchmarks against these key query features systematically (Section 22.4) and point out the weaknesses of current benchmarks that are addressed by FEASIBLE. Our benchmark generation process is presented in Section 22.5. A detailed comparison with DBPSB and an evaluation of the state-of-the-art triple stores follows thereafter. The results are then discussed and we finally conclude. FEASIBLE is open-source and available online at `https://code.google.com/p/feasible/`. A demo can be found at `http://feasible.aksw.org/`.

22.3 PRELIMINARIES

In this section, we define key concepts necessary to understand the subsequent sections of this work. We represent each basic graph pattern (BGP) of a SPARQL query as a directed hypergraph (DH) according to (Saleem and Ngonga Ngomo, 2014). We chose this representation because it allows representing property-property joins, which previous works (Aluç et al., 2014; Görlitz et al., 2012) do not allow to model.

The DH representation of a BGP is formally defined as follows:

Definition 18. *Each basic graph patterns* BGP_i *of a SPARQL query can be represented as a DH* $HG_i = (V, E, \lambda_{vt})$*, where*

- $V = V_s \cup V_p \cup V_o$ *is the set of vertices of* HG_i*,* V_s *is the set of all subjects in* HG_i*,* V_p *the set of all predicates in* HG_i *and* V_o *the set of all objects in* HG_i*;*
- $E = \{e_1, \ldots, e_t\} \subseteq V^3$ *is a set of directed hyperedges (short: edge). Each edge* $e = (v_s, v_p, v_o)$ *emanates from the triple pattern* $<v_s, v_p, v_o>$ *in* BGP_i*. We represent these edges by connecting the head vertex* v_s *with the tail hypervertex* (v_p, v_o)*. We use* $E_{in}(v) \subseteq E$ *and* $E_{out}(v) \subseteq E$ *to denote the set of incoming and outgoing edges of a vertex* v*;*
- λ_{vt} *is a vertex-type-assignment function. Given a vertex* $v \in V$*, its vertex type can be 'star', 'path', 'hybrid', or 'sink' if this vertex participates in at least one join. A 'star' vertex has more than one outgoing edge and no incoming edge. A 'path' vertex has exactly one incoming and one outgoing edge. A 'hybrid' vertex has either more than one incoming and at least one outgoing edge or more than one outgoing and at least one incoming edge. A 'sink' vertex has more than one incoming edge and no outgoing edge. A vertex that does not participate in any join is of type 'simple'.*

Listing 22.1: Examplary SPARQL query

```
SELECT DISTINCT * WHERE
{
        ?drug :description ?drugDesc.
        ?drug :drugType :smallMolecule.
        ?drug :keggCompoundId ?compound.
        ?enzyme :xSubstrate ?compound.
        ?Chemicalreaction :xEnzyme ?enzyme.
        ?Chemicalreaction :equation ?ChemicalEquation.
        ?Chemicalreaction :title ?ReactionTitle .
}
```

The representation of a complete SPARQL query as a DH is the union of the representations of query's BGPs. As an example, the DH representation of the query in Listing 22.1 is shown in Figure 22.1. Based on the DH representation of SPARQL queries we can define the following features of SPARQL queries:

Definition 19 (Number of Triple Patterns)**.** *From Definition 18, the total number of triple patterns in a* BGP_i *is equal to the number of hyperedges* $|E|$ *in the DH representation of the* BGP_i*.*

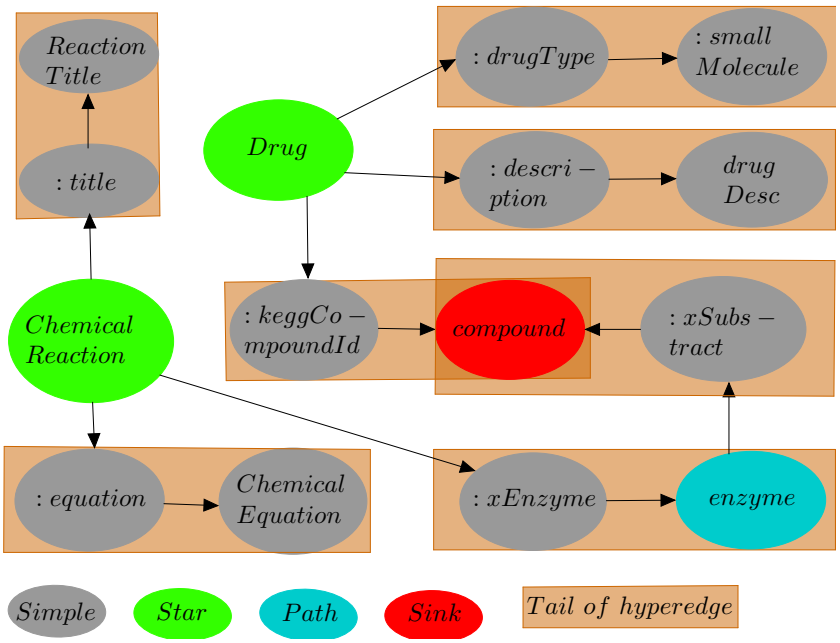


Figure 22.1: Representation of the SPARQL query in Listing 22.1 as directed labelled hypergraph. Prefixes are omitted for simplicity.

Definition 20 (Number of Join Vertices). *Let* $ST = \{st_1, \ldots, st_j\}$ *be the set of vertices of type 'star',* $PT = \{pt_1, \ldots, pt_k\}$ *be the set of vertices of type 'path',* $HB = \{hb_1, \ldots, hb_l\}$ *be the set of vertices of type 'hybrid', and* $SN = \{sn_1, \ldots, sn_m\}$ *be the set of vertices of type 'sink' in a DH representation of a SPARQL query, then the total number of join vertices in the query* $\#JV = |ST| + |PT| + |HB| + |SN|$.

Definition 21 (Join Vertex Degree). *Based on the DH representation of SPARQL queries, the join vertex degree of a vertex* v *is* $JVD(v) = |E_{in}(v)| + |E_{out}(v)|$, *where* $E_{in}(v)$ *resp* $E_{out}(v)$ *is the set of incoming resp. outgoing edges of* v.

Definition 22 (Triple Pattern Selectivity). *Let* tp_i *be a triple pattern and* d *be a* relevant *source for* tp_i. *Furthermore, let* N *be the total number of triples in* d *and* N_m *be the total number of triples in* d *that matches* tp_i, *then the selectivity of* tp_i *w.r.t.* d *is* $Sel(tp_i, d) = N_m/N$.

According to previous works (Aluç et al., 2014; Görlitz et al., 2012), a SPARQL query benchmark should vary the queries it contains w.r.t. the following *query characteristics*: number of triple patterns, number of join vertices, mean join vertex degree, query result set sizes, mean triple pattern selectivities, join vertex types ('star', 'path', 'hybrid', 'sink'), and SPARQL clauses used (e.g., `LIMIT`, `OPTIONAL`, `ORDER BY`, `DISTINCT`, `UNION`, `FILTER`, `REGEX`). In addition, a SPARQL benchmark should contain (or provide options to select) all four SPARQL query forms (i.e., `SELECT`, `DESCRIBE`, `ASK`, and `CONSTRUCT`). Furthermore, the benchmark should contain queries of varying runtimes, ranging from small to reasonably large query execution times. In the next section, we compare state-of-the-art SPARQL benchmarks based on these query features.

		LUBM	BSBM	SP2Bench	WatDiv	DBPSB	F-DBP	DBP	F-SWDF	SWDF
	#Queries	15	125	12	125	125	125	130466	125	64030
Forms (%)	SELECT	100	80	91.67	100	100	95.2	97.9	92.8	58.7
	ASK	0	0	8.33	0	0	0	1.93	2.4	0.09
	CONSTRUCT	0	4	0	0	0	4	0.09	3.2	0.04
	DESCRIBE	0	16	0	0	0	0.8	0.02	1.6	41.1
Clauses (%)	UNION	0	8	16.67	0	36	40.8	7.97	32.8	29.3
	DISTINCT	0	24	41.6	0	100	52.8	4.1	50.4	34.18
	ORDER BY	0	36	16.6	0	0	28.8	0.3	25.6	10.67
	REGEX	0	0	0	0	4	14.4	0.2	16	0.03
	LIMIT	0	36	8.33	0	0	38.4	0.4	45.6	1.79
	OFFSET	0	4	8.33	0	0	18.4	0.03	20.8	0.14
	OPTIONAL	0	52	25	0	32	30.4	20.1	32	29.5
	FILTER	0	52	58.3	0	48	58.4	93.3	29.6	0.72
	GROUP BY	0	0	0	0	0	0.8	7.6×10^{-6}	19.2	1.34
Results	Min	3	0	1	0	197	1	1	1	1
	Max	1.3×10^{4}	31	4.3×10^{7}	4.1×10^{9}	4.6×10^{6}	1.4×10^{6}	1.4×10^{6}	3.0×10^{5}	3.0×10^{5}
	Mean	4.9×10^{3}	8.3	4.5×10^{6}	3.4×10^{7}	3.2×10^{5}	5.2×10^{4}	404	9091	39.5
	S.D.	1.1×10^{4}	9.03	1.3×10^{7}	3.7×10^{8}	9.5×10^{5}	1.9×10^{5}	1.2×10^{4}	4.7×10^{4}	2208
BGPs	Min	1	1	1	1	1	1	0	0	0
	Max	1	5	3	1	9	14	14	14	14
	Mean	1	2.8	1.5	1	2.69	3.17	1.67	2.68	2.28
	S.D.	0	1.70	0.67	0	2.43	3.55	1.66	2.81	2.9
TPs	Min	1	1	1	1	1	1	0	0	0
	Max	6	15	13	12	12	18	18	14	14
	Mean	3	9.32	5.9	5.3	4.5	4.8	1.7	3.2	2.5
	S.D.	1.81	5.17	3.82	2.60	2.79	4.39	1.68	2.76	3.21
JV	Min	0	0	0	0	0	0	0	0	0
	Max	4	6	10	5	3	11	11	3	3
	Mean	1.6	2.88	4.25	1.77	1.21	1.29	0.02	0.52	0.18
	S.D.	1.40	1.80	3.79	0.99	1.12	2.39	0.23	0.65	0.45
MJVD	Min	0	0	0	0	0	0	0	0	0
	Max	5	4.5	9	7	5	11	11	4	5
	Mean	2.02	3.05	2.41	3.62	1.82	1.44	0.04	0.96	0.37
	S.D.	1.29	1.63	2.26	1.40	1.43	2.13	0.33	1.09	0.87
MTPS	Min	3.2×10^{-4}	9.4×10^{-8}	6.5×10^{-5}	0	1.1×10^{-5}	2.8×10^{-9}	1.2×10^{-5}	1.0×10^{-5}	1.0×10^{-5}
	Max	0.432	0.045	0.53	0.011	1	1	1	1	1
	Mean	0.01	0.01	0.22	0.004	0.119	0.140	0.005	0.291	0.0238
	S.D.	0.074	0.01	0.20	0.002	0.22	0.31	0.03	0.32	0.07
Runtime	Min	2	5	7	3	11	2	1	4	3
	Max	3200	99	7.1×10^{5}	8.8×10^{8}	5.4×10^{4}	3.2×10^{4}	5.6×10^{4}	4.1×10^{4}	4.1×10^{4}
	Mean	437	9.1	2.8×10^{5}	4.4×10^{8}	1.0×10^{4}	2242	30.4	1308	16.1
	S.D.	320	14.5	5.2×10^{5}	2.7×10^{7}	1.7×10^{4}	6961	702.5	5335	249.6

Table 22.1: Comparison of SPARQL benchmarks and query logs (**F-DBP** = FEASIBLE Benchmarks from DBpedia query log, **DBP** = DBpedia query log, **F-SWDF** = FEASIBLE Benchmark from Semantic Web Dog Food query log, **SWDF** = Semantic Web Dog Food query log, **TPs** = Triple Patterns, **JV** = Join Vertices, **MJVD** = Mean Join Vertices Degree, **MTPS** = Mean Triple Pattern Selectivity, **S.D.** = Standard Deviation). Runtime(ms)

22.4 A COMPARISON OF EXISTING BENCHMARKS AND QUERY LOGS

Different benchmarks have been proposed to compare triple stores for their query execution capabilities and performance. Table 22.1 provides a detailed summary of the characteristics of the most commonly used benchmarks as well as of two real query logs. All benchmark executions and result set computations were carried out on a machine with 16 GB RAM and a 6-Core i7 3.40 GHz CPU running Ubuntu 14.04.2. All synthetic benchmarks were configured to generate 10 million triples. We ran LUBM (Guo et al., 2005) on OWLIM-Lite as it requires reasoning. All other benchmarks were ran on virtuoso 7.2 with NumberOfBuffers = 1360000, and MaxDirtyBuffers = 1000000. As query logs, we used (1) the portion of the DBpedia 3.5.1 query log (a total of 3,159,812 queries) collected between April 30th, 2010 and July 20th, 2010[1] as well as (2) the portion of the Semantic Web Dog Food (SWDF) query log (a total of 1,414,391 queries) gathered between May 16th, 2014 and November 12th, 2014. Note that we only considered queries (called *cleaned queries*) which produce at least 1 result after the query execution (130,466 queries from DBpedia and 64,029 queries from SWDF).[2] In the following, we compare these benchmarks and query logs w.r.t. the features shown in Table 22.1.

LUBM was designed to test the triple stores and reasoners for their reasoning capabilities. It is based on a customizable and deterministic generator for synthetic data. The queries included in this benchmark commonly lead to query results sizes ranges from 2 to 3200, query triple patterns ranges from 1 to 6, and all the queries consist of a single BGP. LUBM includes a fixed number of `SELECT` queries (i.e., 15) where none of the clauses shown in Table 22.1 is used.

The Berlin SPARQL Benchmark (BSBM) (Bizer and Schultz, 2009) uses a total of 125 query templates to generate any number of SPARQL queries for benchmarking. Multiple use cases such as explore, update, and business intelligence are included in this benchmark. Furthermore, it also includes many of the important SPARQL clauses of Table 22.1. However, the queries included in this benchmark are rather simple with an average query runtime of 9.1 ms and a largest query result set size of 31.

SP^{2}Bench mirrors vital characteristics (such as power law distributions or Gaussian curves) of the data in the DBLP bibliographic database. The queries given in benchmark are mostly complex. For example, the mean (across all queries) query result size is above one million and the query runtimes are in the order of 10^5 ms (see Table 22.1).

The Waterloo SPARQL Diversity Test Suite (WatDiv) (Aluç et al., 2014) addresses the limitations of previous benchmarks by providing a synthetic data and query generator to generate large number of queries from a total of 125 queries templates. The queries cover both simple and complex categories with varying number of features such as result set sizes, total number of query triple patterns, join vertices and mean join vertices degree. However, this benchmark is restricted to conjunctive `SELECT` queries (single BGPs). This means that non-conjunctive SPARQL queries (e.g., queries which make use of the `UNION` and `OPTIONAL` features) are not considered. Furthermore, WatDiv does not consider other important SPARQL clauses, e.g., `FILTER` and `REGEX`. However, our analysis of the query logs of DBpedia3.5.1

1 We chose this query log because it was used by DBPSB.

2 The datadumps, query logs and cleaned queries for both datasets can be downloaded from project home page

and SWDF given in Table 22.1 shows that 20.1% resp. 7.97% of the DBpedia queries make use of `OPTIONAL` resp. `UNION` clauses. Similarly, 29.5% resp. 29.3% of the SWDF queries contain `OPTIONAL` resp. `UNION` clauses.

While the distribution of query features in the benchmarks presented so far is mostly static, the use of different SPARQL clauses and triple pattern join types varies greatly from dataset to dataset, thus making it very difficult for any synthetic query generator to reflect real queries. For example, the DBpedia and SWDF query log differ significantly in their use of `DESCRIBE` (41.1% for SWDF vs 0.02% for DBpedia), `FILTER` (0.72% for SWDF vs 93.3% for DBpedia) and `UNION` (29.3% for SWDF vs 7.97% for DBpedia) clauses. Similar variations have been reported in (Picalausa and Vansummeren, 2011) as well. To address this issue, the DBpedia SPARQL Benchmark (DBPSB) (Morsey et al., 2011, 2012) (which generates benchmark queries from query logs) was proposed. However, this benchmark does not consider key query features (i.e., number of join vertices, mean join vertices degree, mean triple pattern selectivities, the query result size and overall query runtimes) while selecting query templates. Note that previous works (Aluç et al., 2014; Görlitz et al., 2012) pointed that these query features greatly affect the triple stores performance and thus should be considered while designing SPARQL benchmarks.

In this work we present FEASIBLE, a benchmark generation framework which is able to generate a customizable benchmark from any set of queries, esp. from query logs. FEASIBLE addresses the drawbacks on previous benchmark generation approaches by taking all of the important SPARQL query features of Table 22.1 into consideration when generating benchmarks. In the following, we present our approach in detail.

22.5 FEASIBLE BENCHMARK GENERATION

The benchmark generation behind our approach consists of 3 main steps. The first step is the cleaning step. Thereafter, the features of the queries are normalized. In a final step, we then select a sample of the input queries that reflects the cleaned input queries and return this sample. The sample can be used as seed in template-based benchmark generation approaches such as DBSBM and BSBM.

22.5.1 *Dataset Cleaning*

The aim of the data cleaning step is to remove erroneous and zero-result queries from the set of queries used to generate benchmarks. This step is not of theoretical necessity but leads to practically reliable benchmarks. To clean the input dataset (here query logs), we begin by excluding all syntactically incorrect queries. The syntactically correct queries which lead to runtime errors[3] as well as queries which return zero results are removed from the set of relevant queries for benchmarking. We attach all 9 SPARQL clauses (e.g., `UNION`, `DISTINCT`) and 7 query features (i.e., runtime, join vertices, etc.) given in Table 22.1 to each of the queries. For the sake of simplicity we call these 16 (i.e., 9+7) properties *query features* in the following. All unique queries are then stored in a file[4] and given as input to the next step.

3 The runtime errors were measured using Virtuoso 7.2.
4 A sample file can be found at `http://goo.gl/YUSU9A`

22.5.2 *Normalization of Feature Vectors*

The query selection process of FEASIBLE requires distances between queries to be computed. To ensure that dimensions with high values (e.g., the result set size) do not bias the selection, we normalize the query representations to ensure that all queries are located in a unit hypercube. To this end, each of the queries gathered from the previous step is mapped to a vector of length 16 which stores the corresponding *query features* as follows: For the SPARQL clauses, which are binary (e.g., `UNION` is either used or not used), we store a value 1 if that clause in used in the query. Otherwise we store a 0. All non-binary feature vectors are normalized by dividing their value with the overall maximal value in the dataset. Therewith, we ensure that all entries of the query representations are values between 0 to 1.

22.5.3 *Query Selection*

The query selection process is based on the idea of exemplars used in (Ngonga Ngomo and Auer, 2011) and is shown in Algorithm 22.1. We assume that we are given (1) a number $e \in \mathbb{N}$ of queries to select as benchmark queries as well as (2) a set of queries L with $|L| = n >> e$, where L is the set of all cleaned and normalized queries. The intuition behind our selection approach is to compute an e-sized partition $\mathcal{L} = \{L_1, \ldots, L_e\}$ of L such that (1) the average distance between the points in two different elements of the partition is high and (2) the average distance of points within a partition is small. We can then select the point closest to the average of each L_i (i.e., the medoid of L_i) to be a prototypical example of a query from L and include it into the benchmark generated by FEASIBLE. We implement this intuition formally by (1) selecting e exemplars (i.e., points that represent a portion of the space) that are as far as possible from each other, (2) partitioning L by mapping every point of L to one of these exemplars to compute a partition of the space at hand and (3) selecting the medoid of each of the partitions of space as a query in the benchmark. In the following, we present each of these steps formally. For the sake of clarity, we use the following running example: $L = \{q_1 = [0.2, 0.2], q_2 = [0.5, 0.3], q_3 = [0.8, 0.5], q_4 = [0.9, 0.1], q_5 = [0.5, 0.5]\}$ and assume that we need a benchmark with $e = 2$ queries. Note for the sake of simplicity, we used feature vectors of length 2 instead of 16.

22.5.3.1 *Selection of Exemplars*

We implement an iterative approach to the selection of exemplars (see lines 1-7 of Algorithm 22.1). We begin by finding the average $\tilde{L} = \frac{1}{n} \sum_{q \in L} q$ of all representations of queries $q \in L$. In our example, this point has the coordinates $[0.58, 0.32]$. The first exemplar X_1 is the point of L that is closest to the average and is given by $X_1 = \arg\min_{x \in L} d(\tilde{L}, x)$, where d stands for the Euclidean distance. In our example, this is the query q_2 with a distance of 0.08. We follow an iterative procedure to extending the set $\mathcal{X}$ of all exemplars: We first find $\eta = \arg\max_{y \in L \setminus \mathcal{X}} \left(\sum_{x \in \mathcal{X}} d(x, y) \right)$. η is the point that is furthest away from all exemplars. In our example, that is the query q_4 with a distance of 0.45 from q_2. We then add η to $\mathcal{X}$ and repeat the

Algorithm 22.1 Query Selection Approach

Require: Set of queries L; Size of the benchmark e
Ensure: Benchmark (set of queries) B

1: $\tilde{L} = \frac{1}{|L|} \sum_{q \in L} q$
2: $X_1 = \{\arg\min_{x \in L} d(\tilde{L}, x)\}$
3: $\mathcal{X} = \{X_1\}$
4: **for** $i = 2; i \leqslant e; i{+}{+}$ **do**
5: $\quad X_i = \{\arg\max_{y \in L \setminus \mathcal{X}} d(y, \mathcal{X})\}$
6: $\quad \mathcal{X} = \mathcal{X} \cup \{X_i\}$
7: **end for**
8: $\mathcal{L} = \emptyset$
9: **for** $i = 1; i \leqslant e; i{+}{+}$ **do**
10: $\quad L_i = \{X_i\}$
11: $\quad \mathcal{L} = \mathcal{L} \cup \{L_i\}$
12: **end for**
13: **for** $i = 1; i \leqslant e; i{+}{+}$ **do**
14: $\quad L_i = \{q \in L \setminus \mathcal{X} : X_i = \arg\min_{X \in \mathcal{X}} d(X, q)\}$
15: **end for**
16: $B = \emptyset$
17: **for** $i = 1; i \leqslant e; i{+}{+}$ **do**
18: $\quad \tilde{L_i} = \frac{1}{|L_i|} \sum_{q \in L_i} q$
19: $\quad b_i = \arg\min_{q \in L_i} d(\tilde{L_i}, q)$
20: $\quad B = B \cup \{b_i\}$
21: **end for**
22: **return** B

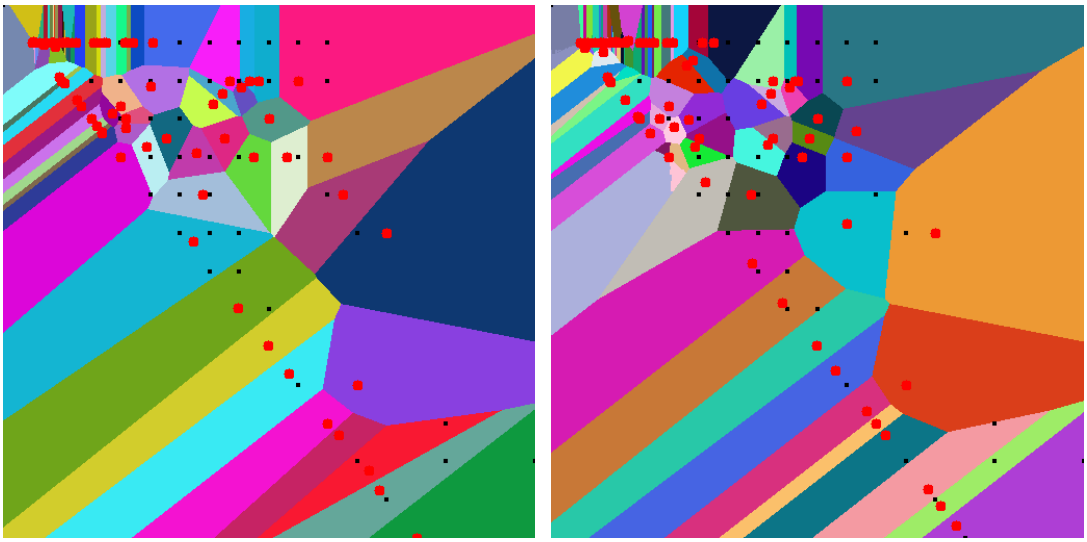
procedure for finding η until $|\mathcal{X}| = e$. Given that $e = 2$ in our example, we get the set $\mathcal{X} = \{q_2, q_4\}$ as set of exemplars.

22.5.3.2 *Selection of Benchmark Queries*

Let $\mathcal{X} = \{X_1, \ldots, X_e\}$ the set of all exemplars. The selection of benchmark queries begins with partitioning the space according to $\mathcal{X}$. The partition L_i is defined as $L_i = \{q \in L : \forall j \neq i : d(q, X_i) \leqslant d(q, X_j)\}$ ((see lines 8-15 of Algorithm 22.1). It is simply the set of queries that are closer to X_i than to any other exemplar. In case of a tie, i.e., $d(q, X_i) = d(q, X_j)$ with $i \neq j$, we assign q to $\min(i, j)$. In our example, we get the following partition: $\mathcal{X} = \{\{q_1, q_2, q_3, q_5\}, \{q_4\}\}$. Finally, we perform the selection of prototypical queries from each partition (see lines 17-22 of Algorithm 22.1). For each partition L_i we begin by computing the average $\tilde{L_i}$ of all representations of queries in L_i. We then select the query $b_i = \arg\min_{q \in L_i} d(\tilde{L_i}, q)$. The set B of benchmark queries is the set of all queries b_i over all L_i. Note that $|B| = e$. In our example, q_4 being the only query in the second partition means that q_4 is selected as representative for the second partition. The average of the first partition is located at $[0.5, 0.375]$. The query q_2 is the closest to the average,

leading to q_2 being selected as representative for the first partition. Our approach thus returns a benchmark with the queries $\{q_2, q_4\}$ as result.

Figure 22.2a and Figure 22.2b show Voronoi diagrams of the results of our approach for benchmarks of size 125 and 175 derived from the DBpedia 3.5.1 query log presented in Table 22.1 along the two dimensions with the highest entropy. Note that some of the queries are superposed in the diagram.



(a) DBpedia-125 (b) DBpedia-175

Figure 22.2: Voronoi diagrams for benchmarks generated by FEASIBLE along the two axes with maximal entropy. Each of the red points is a benchmark query. Several points are superposed as the diagram is a projection of a 16-dimensional space unto 2 dimensions.

22.6 COMPLEXITY ANALYSIS

In the following, we study the complexity of our benchmark generation approach. We denote the number of features considered during the generation process with d. e is the number of exemplars and $|L|$ the size of the input dataset. *Reading and cleaning* the file can be carried out in $O(|L|d)$ as each query is read once and the features are extracted one at a time. We now need to *compute the exemplars*. We begin by computing the average A of all queries, which can be carried out using $O(|L|d)$ arithmetic operations. Finding the query that is nearest to A has the same complexity. The same approach is used to detect the other exemplars, leading to an overall complexity of $O(e|L|d)$ for the computation of exemplars. *Mapping each point* to the nearest exemplar has an a-priori complexity of $O(e|L|d)$ arithmetic operations. Given that the distances between the exemplars and all the points in L are available from the previous step, we can simply look up the distances and thus gather this information in $O(1)$ for each pair of exemplar and point, leading to an overall complexity of $O(e|L|)$. Finally, *the selection of the representative in the cluster* demands averaging the elements of the cluster and selecting the query that is closest to this point. For each cluster of size $|C_l|$, we need $(d|C_l|)$ arithmetic operations to find the average point. The holds for finding the query nearest to the average. Given that the sum of the sizes of all the clusters is $|L|$, we can conclude

that the overall complexity of the selection step is $O(d|L|)$. Overall, the worst-case complexity of our algorithm is thus $O(d|L||E|)$.

In the best case, no queries passes the cleaning test, leading to no further processing and to the same complexity as reading the data, which is $O(|L|d)$. The same best-case complexity holds when a benchmark is generated. Here, the filtering step returns exactly e queries, leading to the exemplar generation step being skipped and thus to a complexity of $O(|L|d)$.

22.7 EVALUATION AND RESULTS

Our evaluation comprises two main parts. First, we compare FEASIBLE with DBPSB w.r.t. how well the benchmarks represent the input data. To this end, we use the composite error function defined below. In the second part of our evaluation, we use FEASIBLE benchmarks to compare triple stores w.r.t. their query execution performance.

22.7.1 *Composite Error Estimation*

The benchmarks we generate aim to find typical queries for a given query log. From the point of view of statistics, this is equivalent to computing a subset of a population that has the same characteristics (here mean and standard deviation) as the original population. Thus, we measure the quality of the sampling approach of a benchmark by how much the mean and standard deviation of the features of its queries deviates from that of the query log. We call μ_i resp. σ_i the mean resp. the standard deviation of a given distribution w.r.t. to the i^{th} feature of the said distribution. Let B be a benchmark extracted from a set of queries L. We use two measures to compute the difference between B and L, i.e., the error on the means E_μ and deviations E_σ

$$E_\mu = \frac{1}{k}\sum_{j=1}^{k}(\mu_i(L) - \mu_i(B))^2 \text{ and } E_\sigma = \frac{1}{k}\sum_{j=1}^{k}(\sigma_i(L) - \sigma_i(B))^2. \quad (22.1)$$

We define a composite error estimation E as the harmonic mean of E_μ and E_σ:

$$E = \frac{2E_\mu E_\sigma}{E_\mu + E_\sigma}. \quad (22.2)$$

22.7.2 *Experimental Setup*

DATASETS AND QUERY LOGS: We used the DBpedia 3.5.1 (232.5M triples) and SWDF (294.8K triples) datasets for triple store evaluation. As queries (see Section 22.4), we used 130,466 cleaned queries for DBpedia and 64,029 cleaned queries for SWDF.

BENCHMARKS FOR COMPOSITE ERROR ANALYSIS: In order to compare FEASIBLE with DBPSB, we generated benchmarks of sizes 15, 25, 50, 75, 100, 125, 150, and 175 queries from the DBpedia 3.5.1 query log. Recall this is exactly the same query log used in DBPSB. DBPSB contains a total of 25 query templates derived from 25 real queries. A single query was generated per query template in order to

generate a benchmark of 25 queries. Similarly, 2 queries were generated per query template for a benchmark of 50 queries and so on. The 15-query benchmark of DBPSB was generated from the 25-query benchmark by randomly choosing 15 of the 25 queries. We chose to show results on a 15-query benchmark because LUBM contains 15 queries while SP^{2}Bench contains 12. We also generated benchmarks of the same size (15-175) from SWDF to compare FEASIBLE's composite errors as well as the performance of triple stores across different datasets.

TRIPLE STORES: We used four triple stores in our evaluation: (1) *Virtuoso Open-Source Edition version 7.2* with NumberOfBuffers = 680000, MaxDirtyBuffers = 500000; (2) *Sesame Version 2.7.8* with Tomcat 7 as HTTP interface and native storage layout. We set the spoc, posc, opsc indices to those specified in the native storage configuration. The Java heap size was set to 6GB; (3) *Jena-TDB (Fuseki) Version 2.0* with a Java heap size set to 6GB and (4) *OWLIM-SE Version 6.1* with Tomcat 7.0 as HTTP interface. We set the entity index size to 45,000,000 and enabled the predicate list. The rule set was empty and the Java heap size was set to 6GB. Ergo, we configured all triple stores to use 6GB of memory and used default values otherwise.

BENCHMARKS: Most of the previous evaluations were carried out on `SELECT` queries only (see Table 22.1). Here, beside evaluating the performance of triples stores on `SELECT` evaluation, we also wanted to compare triple stores on the other three forms of SPARQL queries. To this end, we generated DBpedia-ASK-100 (100-ASK-query benchmark derived from DBpedia) and SWDF-ASK-50 (50-ASK-query benchmark derived from SWDF)[5] and compared the selected triple stores for their `ASK` query processing performances. Similarly, we generated DBpedia-CONSTRUCT-100 and SWDF-CONSTRUCT-23, DBpedia-DESCRIBE-25 and SWDF-DESCRIBE-100, and DBpedia-SELECT-100 and SWDF-SELECT-100 benchmarks to test the selected systems for `CONSTRUCT`, `DESCRIBE`, and `SELECT` queries, respectively. Furthermore, we generated DBpedia-Mix-175 (DBpedia benchmark of 175 mix queries of all the four query forms) and SWDF-Mix-175 to test the selected triple stores for their general query processing performance.

BENCHMARK EXECUTION: The evaluation was carried out one triple store at a time on one machine. First, all datasets were loaded into the selected triple store. Once the triple store had completed the data loading, the 2-phase benchmark execution phase began: (1) *Warm-up Phase*: To measure the performance of the triple store under normal operational conditions, a warm-up phase was used where random queries from the query log were posed to triple stores for 10 minutes; (2) *Hot-run Phase:* During this phase, the benchmark query mixes were sent to the tested store. We kept track of the average execution time of each query as well as of the number of query mixes per hour (QMpH). This phase lasted for two hours for each triple store. Note that the benchmark and the triple store were run on the same machine to avoid network latency. We set the query timeout to 180 seconds. The query was aborted after that and maximum time of 180 seconds was used as the query runtime for all queries which timed out. All the data (data dumps, benchmarks, query logs, FEASIBLE code) to repeat our experiments along with complete evaluation results are available at the project website.

5 We chose to select only 50 queries because the SWDF log we used does not contain enough `ASK` queries to generate a 100-query benchmark.

22.7.3 *Experimental Results*

22.7.3.1 *Composite Error*

Table 22.2 shows a comparison of the composite errors of DBPSB and FEASIBLE for different benchmarks. Note that DBPSB queries templates are only available for the DBpedia query log. Thus, we were not able to calculate DBPSB's composite errors for SWDF. As an overall composite error evaluation, FEASIBLE's composite error is 54.9% smaller than DBPSB. The reason for DBPSB's error being higher that FEASIBLE's lies in the fact that it only considers the number of query triple patterns and the SPARQL clauses `UNION`, `OPTIONAL`, `FILTER`, `LANG`, `REGEX`, `STR`, and `DISTINCT` as features. Important query features (such as query result sizes, execution times, triple patterns and join selectivities, and number of join vertices) were not considered when generating the 25 query templates.[6] Furthermore, DBPSB only includes `SELECT` queries. The other three SPARQL query forms, i.e., `CONSTRUCT`, `ASK`, and `DESCRIBE` are not considered. In contrast, our approach considers all of the query forms, SPARQL clauses, and query features reported in Table 22.1.[7] It is important to mention that FEASIBLE's overall composite error across both datasets is only 0.038.

Benchmark	FEASIBLE			DBPSB			Benchmark	FEASIBLE		
	E_μ	E_σ	E	E_μ	E_σ	E		E_μ	E_σ	E
DBpedia-15	0.045	0.054	0.049	0.139	0.192	0.161	SWDF-15	0.019	0.043	0.026
DBpedia-25	0.041	0.054	0.046	0.113	0.139	0.125	SWDF-25	0.034	0.051	0.041
DBpedia-50	0.045	0.056	0.050	0.118	0.132	0.125	SWDF-50	0.036	0.052	0.043
DDBpedia-75	0.053	0.061	0.057	0.096	0.095	0.096	SWDF-75	0.035	0.051	0.042
DDBpedia-100	0.054	0.064	0.059	0.130	0.132	0.131	SWDF-100	0.036	0.050	0.042
DDBpedia-125	0.054	0.064	0.058	0.088	0.082	0.085	SWDF-125	0.034	0.048	0.040
DBpedia-150	0.055	0.064	0.059	0.107	0.124	0.115	SWDF-150	0.033	0.046	0.038
DBpedia-175	0.055	0.065	0.059	0.127	0.144	0.135	SWDF-175	0.033	0.045	0.038
Average	**0.050**	**0.060**	**0.055**	0.115	0.130	0.121	Average	0.032	0.048	0.039

Table 22.2: Comparison of the Mean E_μ , Standard Deviation E_σ and Composite E errors for different benchmark sizes of DBpedia and Semantic Web Dog Food query logs. FEASIBLE outperforms DBPSB across all dimensions.

22.7.3.2 *Triple Store Performance*

Figure 22.3 shows a comparison of the selected triple stores in terms of *queries per second* (QpS) and *query mixes per hour* (QMpH) for different benchmarks generated by FEASIBLE. Table 22.3 shows the overall rank-wise query distributions of the triple stores. Our ranking is partly different from the DBPSB ranking. Overall, (for mix DBpedia and SWDF benchmarks of 175 queries each, Figure 22.3e to Figure 22.3g), Virtuoso ranks first followed by Fuseki, OWLIM-SE, and Sesame. Virtuoso is 59% faster than Fuseki. Fuseki is 1.7% faster than OWLIM-SE, which in turn 16% faster than Sesame.[8]

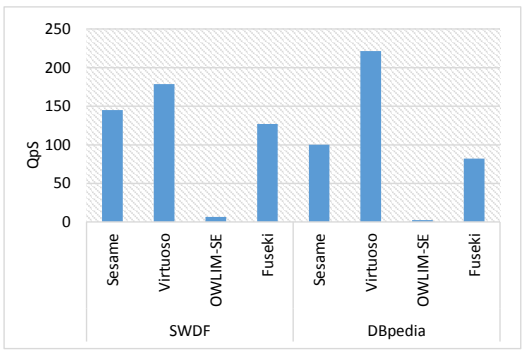
6 Queries templates available at: http://goo.gl/1oZCZY
7 See FEASIBLE online demo for the customization of these features
8 Note the percentage improvements are calculated from the QMpH values as A is (1-QMpH(A)/QMpH(B)*100) percent faster than B.

A more fine-grained look at the evaluation reveals surprising findings: On ASK queries, Virtuoso is clearly faster than the other frameworks (45% faster than Sesame, which is 16% faster than Fuseki, which is in turn 96% faster than OWLIM-SE, see Figure 22.3a). The ranking changes for CONSTRUCT queries: While Virtuoso is still first (87% faster than OWLIM-SE), OWLIM-SE is now faster that 14% faster than Fuseki, which in turn is 42% faster than Sesame (Figure 22.3b). The most drastic change occurs on the DESCRIBE benchmark, where Fuseki ranks first (66% faster than Virtuoso, which is 86% faster than OWLIM-SE, which in turns 47% faster than Sesame, see Figure 22.3c). Yet another ranking emerges from the SELECT benchmarks, where Virtuoso is overall 55% faster than OWLIM-SE, which is 41% faster than Fuseki, which in turns 11% faster than Sesame (Figure 22.3d). These results show that the performance of triple stores varies greatly across the four basic SPARQL forms and none of the system is the sole winner across all query forms. Moreover, the ranking also varies across the different datasets (see, e.g., ASK benchmark for DBpedia and SWDF). Thus, our results suggest that (1) a benchmark should comprise a mix of SPARQL ASK, CONSTRUCT, DESCRIBE, and SELECT queries that reflects the real intended usage of the triple stores to generate accurate results and (2) there is no universal winner amongst triple stores, which points again towards the need to create customized benchmarks for applications when choosing their backend. FEASIBLE addresses both of these requirements by allowing users to generate dedicated benchmarks from their query logs.

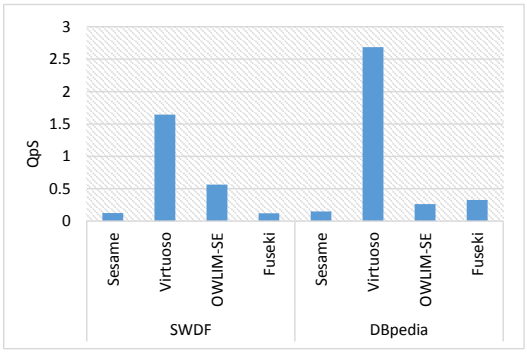
Some interesting observations were revealed by the rank-wise queries distributions of triple stores shown in Table 22.3: First, none of the system is sole winner or loser for a particular rank. Overall, Virtuoso's performance mostly lies in the higher ranks, i.e., rank 1 and 2 (68.29%). This triple store performs especially well on CONSTRUCT queries. Fuseki's performance is mostly in the middle ranks, i.e., rank 2 and 3 (65.14%). In general, it is faster for DESCRIBE queries and is on a slower side for CONSTRUCT and queries containing FILTER and ORDER BY clauses. While OWLIM-SE's performance is usually on the slower side, i.e., rank 3 and 4 (60.86 %), it performs well on complex queries with large result set sizes and complex SPARQL clauses. Finally, Sesame is either fast or slow. For example, for 31.71% of the queries, it achieve the rank 1 (second best after Virtuoso) and but achieves rank 4 on 23.14% of the queries (second worse after OWLIM-SE). In general Sesame is very efficient on simple queries with small result set sizes, a small number of triple triple patterns, and a few SPARQL clauses. However, it performs poorly as soon as the queries grow in complexity. These results shows yet another aspect of the importance of taking structural and data-driven features into consideration while generating benchmarks as they allow deeper insights into the type of queries on which systems perform well or poorly.

	SWDF				DBpedia				Overall			
Triple Store	1st	2nd	3rd	4th	1st	2nd	3rd	4th	1st	2nd	3rd	4th
Virtuoso	38.29	24.57	21.71	15.43	54.86	18.86	15.43	10.86	46.57	21.71	18.57	13.14
Fuseki	17.14	39.43	32.00	11.43	24.00	34.86	24.00	17.14	20.57	37.14	28.00	14.29
OWLIM-SE	10.29	30.29	21.14	38.29	13.14	24.57	25.14	37.14	11.71	27.43	23.14	37.71
Sesame	37.71	12.00	29.14	21.14	25.71	16.57	32.57	25.14	31.71	14.29	30.86	23.14

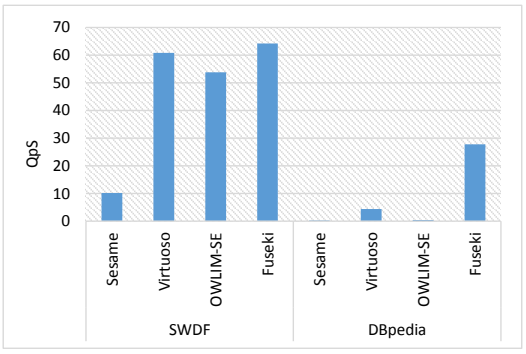
Table 22.3: Overall rank-wise ranking of triple stores. All values are in percentages.



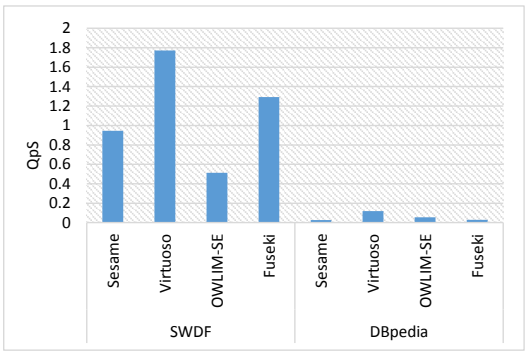
(a) QpS (ASK-Only)



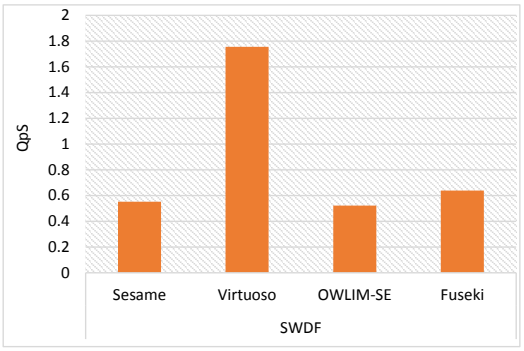
(b) QpS (CONSTRUCT-Only)



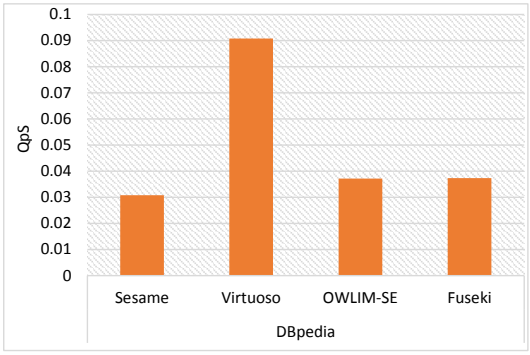
(c) QpS (DESCRIBE-Only)



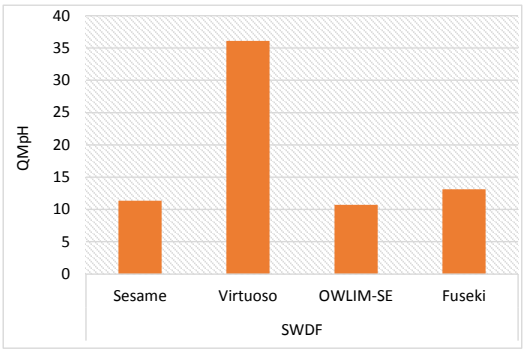
(d) QpS (SELECT-Only)



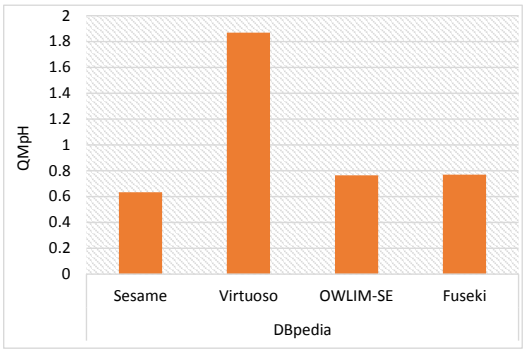
(e) QpS (SWDF-Mix)



(f) QpS (DBpedia-Mix)



(g) QMpH (SWDF-Mix)



(h) QMpH (DBpedia-Mix))

Figure 22.3: Comparison of the triple stores in terms of Queries per Second (QpS) and Query Mix per Hour (QMpH), where a query mix comprises 175 distinct queries

Finally, we also looked into the number of query timeouts during the complete evaluation. Most of the systems time out for `SELECT` queries. Overall, Sesame has the highest number of timeouts (43) followed by Fuseki (32), OWLIM-SE (22), and Virtuoso (14). For Virtuoso, the timeout queries have at least one triple pattern with an unbound subject, an unbound predicate and an unbound object (i.e., a triple pattern of the form `?s ?p ?o`). The corresponding result sets were so large that they could not be computed in 3 minutes. The other three systems mostly timeout for the same queries. OWLIM-SE generally performs better for complex queries with large result set sizes. Fuseki has problems with queries containing `FILTER` (12/32) and `ORDER BY` clauses (11/32 queries). Sesame's performance is slightly worse for complex queries containing many triple patterns and joins as well as complex SPARQL clauses. Note that Sesame also times out for 8 `CONSTRUCT` queries. All the timeout queries for each triple store are provided at the project website.

22.8 CONCLUSION

In this paper we presented FEASIBLE, a customizable SPARQL benchmark generation framework. We compared FEASIBLE with DBPSB and showed that our approach is able to produce high-quality (in terms of their composite error) benchmarks. In addition, our framework allows users to generate customized benchmarks suited for a particular use case, which is of utmost importance when aiming to gather valid insights into the real performance of different triple stores for a given application. This is demonstrated by our triple store evaluation, which shows that the ranking of triple stores varies greatly across different types of queries as well as across datasets. Our results thus suggest that all of the four query forms should be included in the future SPARQL benchmarks. For the sake of future work, we have started converting Linked Data query logs into RDF and made available through the LSQ (Saleem et al., 2015) endpoint. Beside the key queries characteristics discussed in Table 22.1, we have attached many of the SPARQL 1.1 features to each of the query. We will extend FEASIBLE to query the LSQ SPARQL endpoint directly so as to gather queries for the benchmark creation process.

23 DEEP WEB EXTRACTION FOR QUESTION ANSWERING

PREAMBLE

A further application of LIMES lies in the preparation of data for RDF-driven search and question answering engines. This chapter introduces DEQA, a conceptual framework that allows combining state-of-the-art semantic technologies with effective data extraction to answer complex user questions. The content of this chapter is taken from (Lehmann et al., 2012). The author designed and implemented the data integration layer for the question answering engines and co-wrote the paper.

23.1 INTRODUCTION

Answering questions such as "find me a flat to rent close to Oxford University with a garden" is one of the challenges that has haunted the semantic web vision since its inception (Berners-Lee et al., 2001). Question answering has also been the holy grail of search engines, as recently illustrated by both Google and Bing touting "structured data" search and "query answering". Though both of these efforts have made great strides in answering questions about general, factual knowledge, they have fallen short for more transient information such as real estate, tickets, or other product offerings. Vertical search engines and aggregators also fail to address such questions, mostly due to a lack of natural language understanding and limited background knowledge.

This is true even though data extraction and semantic technologies aim to address this challenge from quite opposite directions: On the one hand, the aim of web extraction is to obtain structured knowledge by analyzing web pages. This does not require publishers to make any changes to existing websites, but requires re-engineering the original data used to generate a website. On the other hand, semantic technologies establish the means for publishers to directly provide and process structured information, avoiding errors in extracting ambiguously presented data, but placing a considerable burden on publishers. Despite this chasm in how they approach question answering, neither has succeeded in producing successful solutions for transient, "deep" web data (in contrast to general, Wikipedia-like knowledge and web sites).

In this paper, we show that in this very dichotomy lies the solution to addressing deep web question answering: We present DEQA, a system that allows the easy combination of semantic technologies, data extraction, and natural language processing and demonstrate its ability to answer questions on Oxford's real estate market. The data is extracted from the majority of Oxford's real estate agencies, despite the fact that none publishes semantic (or other structured) representations of their data, and combined with background knowledge, e.g., to correlate real estate offers with points of interest such as the "Ashmolean Museum" or close-by supermarkets.

DEQA is the first comprehensive framework for *deep web question answering* approaching the problem as a combination of three research areas:

- *Web data extraction* – to obtain offers from real estate websites, where no structured interface for the data is available (which happens to be the case for all Oxford real estate agencies).
- *Data integration* – to interlink the extracted data with background knowledge, such as geo-spatial information on relevant points of interest.
- *Question answering* – to supply the user with a natural language interface, capable of understanding even complex queries.

For example a query like "find me a flat to rent close to Oxford University with a garden" can be answered by DEQA. However, this cannot be achieved without adaptation to the specific domain. The unique strength of DEQA is that it is based not only on best-of-breed data extraction, linking, and question answering technology, but also comes with a clear methodology specifying how to adapt DEQA to a specific domain. In Section 23.3, we discuss in detail what is required to adapt DEQA to a new domain and how much effort that is likely to be.

DEQA COMPONENTS We developed DEQA as a conceptual framework for enhancing classic information retrieval and search techniques using recent advances in three technologies for the above problems, developed by the three groups involved in DEQA: DIADEM at Oxford, AKSW at Leipzig, and CITEC at Bielefeld.

- OXPATH is a light-weight data extraction system particularly tailored to quick wrapper generation on modern, scripted web sites. As demonstrated in (Furche et al., 2013), OXPATH is able to solve most data extraction tasks with just four extensions to XPATH, the W3C's standard query language for HTML or XML data. Furthermore, through a sophisticated garbage collection algorithm combined with tight control of the language complexity, OXPATH wrappers outperforms existing data extraction systems by a wide margin (Furche et al., 2013). For the purpose of integration into DEQA, we extended OXPATH with the ability to direct extract RDF data, including type information for both entities and relations.
- The real estate offers extracted with OXPATH contain no or little contextual knowledge, e.g., about general interest locations or typical ranges for the extracted attributes. To that end, we link these extracted offers with external knowledge. This is essential to answer common-sense parts of queries such as "close to Oxford University". Specifically, we employ the LIMES (Ngonga Ngomo and Auer, 2011; Ngonga Ngomo, 2011) framework, which implements time-efficient algorithms for the discovery of domain-specific links to external knowledge bases such as DBpedia (Auer et al., 2007).
- To apply question answering in a straightforward fashion on the supplied, extracted, and enriched knowledge, we employ the *TBSL* approach (Unger et al., 2012) for translating natural language questions into SPARQL queries. TBSL disambiguates entities in the queries and then maps them to templates which capture the semantic structure of the natural language question. This enables the understanding of even complex natural language containing, e.g.,

quantifiers such as the most and more than, comparatives such as higher than and superlatives like the highest – in contrast to most other question answering systems that map natural language input to purely triple-based representations.

Using the combination of these three technologies allows us to adjust to a new domain in a short amount of time (see Section 23.3), yet to answer a significant percentage of questions about real estate offers asked by users (see Section 23.4).

CONTRIBUTIONS. These results validate our hypothesis, that the combination of these technologies can (and may be necessary to) yield accurate question answering for a broad set of queries in a specific domain. This is achieved without requiring publishers to provide structured data and at a fairly low effort for domain adaptation. Specifically,

1. DEQA is the first comprehensive *deep web question answering system for entire domains* that can answer the majority of natural language questions about objects only available in form of plain, old HTML websites (Section 23.2).
2. These websites are turned into structured RDF data through an *extension of* OXPATH *for RDF output*, providing a concise syntax to extract object and data properties (Section 23.2.1).
3. By extracting this data into RDF and *linking* it with background knowledge, it can answer not only queries for specific attributes ("in postcode OX1"), but also queries using *common-sense criteria* ("close to Oxford University"), see Section 23.2.2.
4. With TBSL, we are able to map such queries to SPARQL expressions even if they include *complex natural language expressions* such as "higher than".
5. DEQA provides a *methodology and framework* that can be rapidly instantiated for new domains, as discussed in Section 23.3.
6. As a *case study*, we instantiate DEQA to Oxford's entire real estate market, involving the 20 largest real estate agents and all of their properties on sale, and illustrate the necessary effort.
7. A user-centric *evaluation* demonstrates that DEQA is able to answer many of the natural language questions asked by users (Section 23.4).

With these contributions, DEQA is the first comprehensive framework for deep web query answering, covering the extraction and data collection process as well as the actual query answering, as elaborated in Section 23.5.

23.2 APPROACH

The overall approach of DEQA is illustrated in Figure 23.1: Given a particular domain, such as real estate, the first step consists of identifying relevant websites and extracting data from those. This previously tedious task can now be reduced to the rapid creation of OXPATH wrappers as described in Section 23.2.1. In DEQA, data integration is performed through a triple store using a common base ontology. Hence, the first phase may be a combination of the extraction of unstructured and

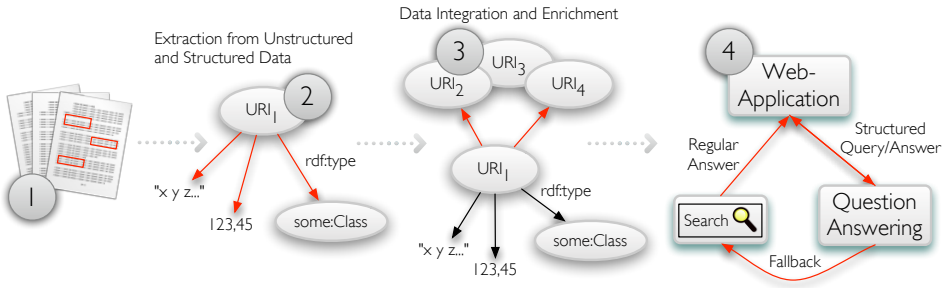


Figure 23.1: Overview of the DEQA conceptual framework

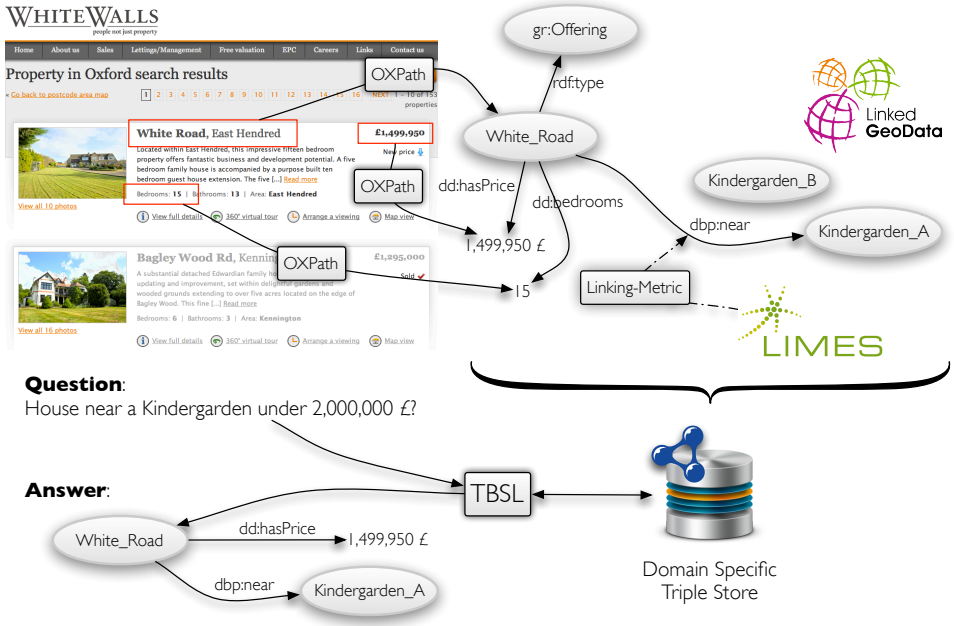


Figure 23.2: Implementation of DEQA for the real-estate domain

structured data. For instance, websites may already expose data as *RDFa*, which can then be transformed to the target schema, e.g. using *R2R* (Bizer and Schultz, 2010), if necessary. This basic RDF data is enriched, e.g. via linking, schema enrichment (Lehmann et al., 2011; Bühmann and Lehmann, 2012), geo-coding or post-processing steps on the extracted data. This is particularly interesting, since the LOD cloud contains a wealth of information across different domains which allows users to formulate queries in a more natural way (e.g., using landmarks rather than postcodes or coordinates). For instance, in our analysis of the real estate domain, over 100k triples for 2,400 properties were extracted and enriched by over 100k links to the Linked Data Cloud. Finally, question answering or semantic search systems can be deployed on top of the created knowledge. One of the most promising research areas in question answering in the past years is the conversion of natural language to SPARQL queries (Unger et al., 2012; Lopez et al., 2011; Unger and Cimiano, 2011), which allows a direct deployment of such systems on top of a triple store. Finally, DEQA first attempts to convert a natural language query to SPARQL, yet can fall back to standard information retrieval, where this fails.

```
1  doc("http://wwagency.co.uk/")//input[@name='search']/{click/}/
2  (descendant::a[@class='pagenum']
3  [text()='NEXT'][1]/{click[wait=1]/})*
4  /descendant::div.proplist_wrap:<(gr:Offering)>
5  [?.//span.prop_price:<dd:hasPrice(xsd:double)=
6  substring-after(.,'$\color{darkorange}\pounds$')>]
7  [?.//a[@class='link_fulldetails']:<foaf:page=string(@href)>]
8  [?.:<gr:includes(dd:House)>
9  [?.//h2:<gr:name=string(.)>]
10 [?.//h2:<vcard:street_address=string(.)>]
11 [?.//div.prop_maininfo//strong[1]:<dd:bedrooms=string(.)>]
12 [? .//img:<foaf:depiction=string(@src)>]
```

Figure 23.3: OxPath RDF wrapper

The domain-specific implementation of the conceptual framework, which we used for the real estate domain, is depicted in Figure 23.2. It covers the above described steps by employing state-of-the-art tools in the respective areas, OxPath for data extraction to RDF, Limes for linking to the Linked Data Cloud, and Tbsl for translating natural language questions to SPARQL queries. In the following, we briefly discuss how each of these challenges are addressed in Deqa.

23.2.1 OxPath *for RDF extraction*

OxPath is a recently introduced (Furche et al., 2013) modern wrapper language that combines ease-of-use (through a very small extension of standard XPath and a suite of visual tools (Kranzdorf et al., 2012)) with highly efficient data extraction. Here, we illustrate OxPath through a sample wrapper shown in Figure 23.3.

This wrapper directly produces RDF triples, for which we extended OxPath with *RDF extraction markers* that generate both data and object properties including proper type information and object identities. For example the extraction markers `<:(gr:Offering>` and `<gr:includes(dd:House)>` in Figure 23.3 produce – given a suitable page – a set of matches typed as `gr:Offering`, each with a set of `dd:House` children. When this expression is evaluated for RDF output, each pair of such matches generates two RDF instances related by `gr:includes` and typed as above (i.e., three RDF triples).

To give a more detailed impression of an OxPath RDF wrapper assuming some familiarity with XPath, we discuss the main features of Figure 23.3:

(Line 1) We first load the web site `wwagency.co.uk`, a real estate agency serving the Oxford area, and click on their search button without restricting the search results. Therein, `{click/}` is an *action* which clicks on all elements in the current context set, in this case, containing only the search button. This action is *absolute*, i.e., after executing the action, OxPath continues its evaluation at the root of the newly loaded document.

(Lines 2–3) Next, we iterate through the next links connecting the paginated results. To this end, we repeat within a *Kleene star* expression the following steps: we select the first link which is of class 'pagenum' and contains the text 'NEXT'. The expression then clicks on the link in an absolute action `{click[wait=1]/}` and waits for a second after the `onload` event to ensure that the heavily scripted page finishes its initialization.

(Line 4) On each result page, we select all `div` nodes of class `proplist_wrap` and extract an `gr:Offering` instance for each such node. Aside from the CSS-like shorthand for classes (analogously, we provide the `#` notation for ids), this subexpression uses the first *RDF extraction marker:* This extraction marker `:<gr:Offering>` produces an *object instance,* because it does not extract a value necessary to produce a *data property.* The remainder of the expression adds object and data properties to this instance, detailing the offering specifics.

(Lines 5–6) We extract the price of the offering by selecting and extracting the `span` of class `prop_price` within the offering `div`. In particular, the marker `:<dd:hasPrice(xsd:double)=substring-after(.,'$\color{darkorange}\pounds$')>` specifies the extraction of a `dd:hasPrice` data property of type `xsd:double` with the value stated after the `'$\color{darkorange}\pounds$'` character. The nesting of RDF properties follows the predicate nesting structure, and thus, as the price is extracted inside a predicate following the extracted offering, this price is associated with the offering. We use an *optional predicate,* `[?$\phi$]`, to ensure that the evaluation continues, even if an offering does not name a price and the predicate extraction fails.

(Line 7) Links to details pages are extracted as `foaf:page` data properties.

(Lines 8–12) Aside having a price, an offering also needs to refer to a property, extracted next. In Line 8, with `:<gr:includes(dd:House)>`, we extract an instance of the `dd:House` class as *object property* of the previous offering (because of the predicate nesting), related via `gr:include`. The remaining four lines extract the name, address, the number of bedrooms, and the property images as data properties belonging to the `dd:House` instance, as all those properties are extracted within nested predicates.

This wrapper produces RDF triples as below, describing two instances, the first one `dd:g31g111` representing a house with 4 bedrooms in Marston, and the second one `dd:g31g109` representing an offer on this house at GBP 475000.

```
1 dd:g31g111
2 a dd:House ;            dd:bedrooms 4 ;
3 gr:name "William Street,  Marston OX3" ;
4 vcard:street-address "William Street,  Marston OX3" ;
5 foaf:depiction "http://www.wwagency.com/i_up/111_1299510028.jpg" .
6 dd:g31g109
7 a gr:offering ;       dd:hasPrice "475000"^^xsd:double ;
8 gr:includes dd:g31g111 .
```

For more details on OxPath, please refer to (Furche et al., 2013). We also provide the full set of wrappers on the project home page.

23.2.2 Limes

We discuss the Limes specification used to link and integrate the RDF data extracted by OxPath with *LinkedGeoData* – a vast knowledge base extracted from OpenStreetMaps containing spatial data including points-of-interest such as schools. The following listing shows an excerpt of the specification that links houses extracted by OxPath with nearby schools. Every link discovery process requires a set S of source and T target instances that are to be linked. In Limes, these can be defined by specifying the *restrictions* on the instances as well as the set of *properties* that these instances must possess to be linked. In our example, the set S (specified

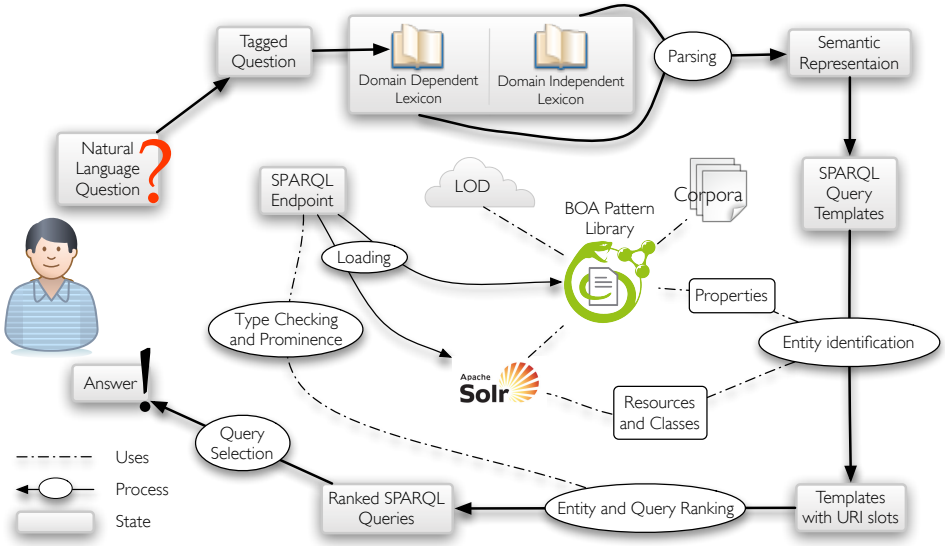


Figure 23.4: Overview of the TBSL question answering engine (source: (Unger et al., 2012)).

by the tag <SOURCE>) consists of `oxford:House` which possess a longitude and a latitude. Similarly, the set T (which is omitted in the listing for brevity) was defined as all the schools whose latitude lies between 50 and 52 degrees and whose longitude lies between -2 and -1 degrees. For instances $a \in S$ and $b \in T$, the similarity is set to

$$\frac{1}{1+\sqrt{(a.\texttt{wgs:lat}-b.\texttt{wgs:lat})^2+(a.\texttt{wgs:long}-b.\texttt{wgs:long})^2)}}. \tag{23.1}$$

Two instances are then considered close to each other (described by the predicate `dbp:near`) if their similarity was at least 0.95.

```
<SOURCE> <ID>oxford</ID>
...
<VAR>?a</VAR>
<RESTRICTION>?a a oxford:House</RESTRICTION>
<PROPERTY>wgs:lat AS number</PROPERTY>
<PROPERTY>wgs:long AS number</PROPERTY> </SOURCE>
...
<METRIC>euclidean(a.wgs:lat|wgs:long, b.wgs:lat|wgs:long)</METRIC>
<ACCEPTANCE> <THRESHOLD>0.9975</THRESHOLD>
<FILE>allNear.ttl</FILE>
<RELATION>dbp:near</RELATION> </ACCEPTANCE>
...
```

The property values of all schools from LinkedGeoData that were found to be close to houses extracted by OxPath were subsequently retrieved by Limes and loaded into the Deqa triple store.

23.2.3 *TBSL Question Answering*

Figure 23.4 gives an overview of our template-based question answering approach TBSL (Unger et al., 2012). The system takes a natural language question as input and returns a SPARQL query and the corresponding answer(s) as output. First,

the natural language question is parsed on the basis of its part-of-speech tags and a domain-independent grammar comprising for example wh-words, determiners, and numerals. The result is a semantic representation of the natural language query, which is then converted into a SPARQL query template. This template fixes the overall structure of the target query, including aggregation functions such as filters and counts, but leaves open slots that still need to be filled with URIs corresponding to the natural language expressions in the input question. For example, the question "Give me all flats near Oxford University" yields the following template query, which contains a class slot for some URI corresponding to "flats", a resource slot for some URI corresponding to "Oxford University", and a property slot that expresses the "near" relation:

```
SELECT ?y WHERE
{ ?y ?p1 ?y0.
?y rdf:type ?p0. }
```

- `y0`: "Oxford University" (resource)
- `p0`: "flats" (class)
- `p1`: "near" (object property)

In order to fill these slots, entity identification approaches are used to obtain appropriate URIs, relying both on string similarity and natural language patterns compiled from existing structured data in the Linked Data Cloud and text documents (cf. (Gerber and Ngonga Ngomo, 2012)). This yields a range of query candidates as potential translations of the input question. Those candidates are ranked on the basis of string similarity values, prominence values, and schema conformance checks. The highest ranked queries are then tested against the underlying triple store and the best answer is returned to the user.

23.3 DOMAIN ADAPTION COSTS

DEQA requires instantiation for a specific domain, however, through advances in semantic and web extraction technologies this adaptation involves far less efforts than in the past and is now feasible even with limited resources. We substantiate this claim by discussing the resources required for our case study on Oxford's real estate for

- system setup and domain adaptation and for and
- maintaining the wrappers and links to background knowledge.

The first step in adapting DEQA to a new domain is the *creation or adaption of a suitable domain ontology* in RDFS. In our case, the ontology consists of 5 object properties, 7 data properties, 9 classes, and 10 individuals, all specified in less than 150 lines of turtle code. We were cautious to capture all relevant cases. Hence we build the ontology iteratively while fitting a dozen representative offers from 4 different agencies into the ontology – reaching already a saturation. The entire process of ontology creation took four domain experts a couple of hours.

Web extraction. Having the ontology, we need to develop wrappers to extract the data from the relevant sites. The process consists of identifying the relevant DOM

features to frame the data to be extracted, and running sufficiently many tests to check the wrapper's behavior on other pages from the same site. The wrappers we employ in our case study took on average 10 minutes each to create, such that it took an OxPath expert less than a day to identify the 20 most relevant web sites and write appropriate wrappers. To ease OxPath wrapper generation, we relied on Visual OxPath (Kranzdorf et al., 2012), a supervised wrapper induction system that generates highly robust wrappers from few examples: The system embeds a real browser, and records user interaction on the page (e.g., navigation, click, form filling). Once, the relevant data has been reached, the user marks the data to extract (e.g., price), and checks whether Visual OxPath generalizes the selection correctly, in case refining the selection. In our user study (Kranzdorf et al., 2012), we show that even users without prior knowledge of OxPath can create a wrapper in less than three minutes (not counting testing and verification) on average.

Linking. Creating Limes link specifications can be carried out in manifold ways. For example, Limes provides active learning algorithms for the semi-automatic detection of link specifications that have been shown to require only a small number of annotations (i.e., $10-40$ depending on the data quality) to detect high-quality link specifications (Ngonga Ngomo et al., 2011; Ngonga Ngomo and Lyko, 2012). Given that we had clear definition of the two predicates `near` (for distances up to 2km) and `atWalkingDistance` (for distances up to 500m) to be computed for the domain at hand, we chose to create link specifications manually for each of these predicates.

Question Answering The component for parsing a user question and constructing query templates requires only little domain adaptation. The core part of the lexicon that is used for parsing comprises domain-independent expressions that can be re-used, all other entries are built on the fly. The only part that was added for Deqa were lexical entries covering some typical tokens with fixed mappings to URIs in the given domain, e.g. "near". This has been done for six mappings, resulting in 20 domain-specific entries. The required manual effort amounts to less than an hour.

System Maintenance

The frequency to which a wrapper needs to be updated is directly correlated to its robustness. Robustness measures the degree of a wrapper to still select the same nodes after changes on the page. Both (Kranzdorf et al., 2012; Gulhane et al., 2011) show that wrappers without robustness consideration have limited lifespan, but Visual OxPath implements a number of techniques to prolong the fitness of its wrappers. In particular, given only a single example, Visual OxPath suggests a list of expressions ranked by robustness of the generated wrapper. We have evaluated the top-ranked suggested wrappers over 26 weeks, showing that they fail only after 26 weeks in contrast to average wrappers that fail in $9-12$ weeks. In Oxford real estate, we estimate that wrapper maintenance will involve about one failing wrapper per week.

Linking and Question Answering The system maintenance for the Limes link specifications is minimal. If the schema is not altered, the specifications created can simply be reran when the data endpoints are updated. In case of an alteration of the schema, the `PROPERTY` and `METRIC` tags of the specification need to be altered. This is yet a matter of minutes if the schema of both endpoints is known. If the new

schema is not known, then the link specification has to be learned anew. Previous work (Ngonga Ngomo and Lyko, 2012) has shown that even on large datasets, learning such a specification requires only about 5 min. For question answering, no regular maintenance effort is usually required. An exception, both for linking and question answering, are schema changes. Such changes can in rare cases invalidate specifications, in which case they have to be altered manually. TBSL is flexible in terms of schema changes as long as entities use appropriate labels or URIs. For instance, in (Unger et al., 2012) was applied to the DBpedia ontology with hundreds of classes and properties without requiring manual configuration for adapting it to this schema. However, previously manually added domain-specific configuration entries for improving the performance of TBSL may require updates in case of schema changes.

23.4 EVALUATION

The components comprising the DEQA platform have been evaluated in the respective reference articles, in particular (Furche et al., 2013) for OXPATH, (Ngonga Ngomo and Auer, 2011) for LIMES, and (Unger et al., 2012) for TBSL. Hence, we are mostly interested in an evaluation of the overall system, as well as specific observation for the Oxford real estate case study. The main benefit of DEQA is to enhance existing search functionality with question answering. Therefore, we evaluate the overall system for the real-estate domain by letting users ask queries and then verifying the results.

First, DEQA was instantiated for Oxford real-estate as described in Section 23.3. The OXPATH wrappers, the LIMES specs and the TBSL configuration are all publicly available at `http://aksw.org/projects/DEQA`. Our dataset consists of more than 2400 offers on houses in Oxfordshire, extracted from the 20 most popular real estate agencies in the area. The wrappers extract spatial information from 50% of the agencies, typically extracted from map links. For all properties in our dataset, we extract street address and locality. The full postcode (e.g., OX27PS) is available in 20% of the cases (otherwise only the postcode area, e.g., OX1 for Oxford central is available). 96% of all offers expose directly the price, the remaining 4% are "price on inquiry". Images and textual descriptions are available for all properties, but not all agencies publish the number of bathrooms, bedrooms and reception rooms. These offers are enriched by LIMES with 93,500 links to near (within 2 kilometres) and 7,500 links to very near (within 500 metres) spatial objects. The data is also enriched by loading 52,500 triples from LinkedGeoData describing the linked objects. Domain specific spatial mappings were added to TBSL, e.g. "walking distance" is mapped to "very near".

We asked 5 Oxford residents to provide 20 questions each. They were told to enter questions, which would typically arise when searching for a new flat or house in Oxford. We then checked, whether the questions could be parsed by TBSL, whether they could successfully be converted to a SPARQL query on the underlying data and whether those SPARQL queries are correct.

23.4.1 *Results and Discussion*

It turned out that most questions would be impossible to answer by only employing information retrieval on the descriptions of properties in Oxford. Many

number of questions	100
—SPARQL queries created	71
—SPARQL queries returning results	63
—SPARQL queries with correct results	49
—exactly intended SPARQL query	30
—SPARQL queries with incorrect results	14

(a) Evaluation results

failures	
—data coverage	9
—linguistic coverage	18
—POS tagging	2
—other reasons	6

(b) Failure reasons

Table 23.1: Evaluation results and failures

questions would also not be possible to answer via search forms on the respective real-estate websites, as they only provide basic attributes (price, bedroom numbers), but neither more advanced ones (such as "Edwardian", with garden) nor have a concept of close-by information (such as close to a supermarket). Even if they can be answered there, the coverage would be low as we extracted data using over 20 wrappers. While some questions had similar structures, there is little overlap in general.

The results of our experiment are shown in 23.1a and 23.1b. Most questions can be converted successfully to SPARQL queries and many of those are the SPARQL queries intended by users of the system. Hence, DEQA provides significant added value in the real estate domain in Oxford despite the relatively small effort necessary for setting up the system. For the questions, which were not correctly answered, we analysed the reasons for failure and summarize them in 23.1b. If questions were not correctly phrased, such as "house with immediately available", they lead to part-of-speech tagging problems and parse failure. Such issues will be dealt with by integration query cleaning approaches into DEQA. In some cases TBSL could not answer the question because it lacks certain features, e.g. negation such as "not in Marston" or aggregates such as average prices in some area. But since TBSL uses a first order logical representation of the input query internally, those features can be added to the QA engine in the future. Support for some aggregates such as COUNT already exists. In some cases, on the other hand, data was insufficient, e.g. users asking for data that was neither extracted by OXPATH nor available through the links to LinkedGeoData, e.g. "house in a corner or end-of-terrace plot". Moreover, some questions contain vague, subjective criteria such as "cheap", "recently" or even "representative", the exact meaning of which heavily depends on the user's reference values. In principle, such predicates could be incorporated in TBSL by mapping them to specific restrictions, e.g. cheap could be mapped to costs for flats less than 800 pounds per month. The extended version of DEQA will be compared with classical retrieval engines to quantify the added value of our approach.

An Example for a successful query is shown in Listing 23.1.

Listing 23.1: Query for "all houses in Abingdon with more than two bedrooms"

```
SELECT ?y WHERE {
  ?y a <http://diadem.cs.ox.ac.uk/ontologies/real-estate#House> .
  ?y <http://diadem.cs.ox.ac.uk/ontologies/real-estate#bedrooms> ?y0 .
  ?y <http://www.w3.org/2006/vcard/ns#street-address> ?y1 .
```

```
  FILTER(?y0 > 2) .
  FILTER(regex(?y1,'Abingdon','i')) .
}
```

In that case, TBSL first performs a restriction by class ("House"), then it finds the town name "Abingdon" from the street address and it performs a filter on the number of rooms. Note that many QA systems over structered data rely on purely triple-based representations (e.g. PowerAqua (Lopez et al., 2009)) and therefore fail to include such filters.

Another example is "Edwardian houses close to supermarket for less than 1 000 000 in Oxfordshire" as shown in Listing 23.2.

Listing 23.2: Query for "Edwardian houses close to supermarket for less than 1 000 000 in Oxfordshire"

```
SELECT ?x0 WHERE {
  ?x0 <http://dbpedia.org/property/near> ?y2 .
  ?x0 a <http://diadem.cs.ox.ac.uk/ontologies/real-estate#House> .
  ?v <http://purl.org/goodrelations/v1#includes> ?x0 .
  ?x0 <http://www.w3.org/2006/vcard/ns#street-address> ?y0 .
  ?v <http://diadem.cs.ox.ac.uk/ontologies/real-estate#hasPrice> ?y1 .
  ?y2 a <http://linkedgeodata.org/ontology/Supermarket> .
  ?x0 <http://purl.org/goodrelations/v1#description> ?y .
  FILTER(regex(?y0,'Oxfordshire','i')) .
  FILTER(regex(?y,'Edwardian ','i')) .
  FILTER(?y1 < 1000000) .
}
```

In that case, the links to LinkedGeoData were used by selecting the "near" property as well as by finding the correct class from the LinkedGeoData ontology.

23.4.2 *Performance Evaluation*

We conclude this evaluation with a brief look at the system performance, focusing on the resource intensive background extraction and linking, which require several hours compared to seconds for the actual query evaluation. For the real-estate scenario, the TBSL algorithm requires 7 seconds on average for answering a natural language query using a remote triple store as backend. The performance is quite stable even for complex queries, which required at most 10 seconds. So far, the TBSL system has not been heavily optimised in terms of performance, since the research focus was clearly to have a very flexible, robust and accurate algorithm. Performance could be improved, e.g., by using fulltext indices for speeding up NLP tasks and queries.

Extraction. In (Furche et al., 2013) we show that the memory requirements of OxPath are independent of the number of pages visited: For DEQA, the average execution time of our wrappers amounts to approximately 30 pages/min. As we do not want to overtax the agencies' websites, this rate is high enough to crawl an entire website in few minutes. For OxPath this rate is quite slow, but is rooted in inherent characteristics of the domain:

- Many real estate websites are *unable to serve requests at higher rates*, and
- supply *heavily scripted pages*, containing many images or fancy features like flash galleries.

Indeed, the evaluation of OxPath is dominated by the browser initialisation and rendering time (Furche et al., 2013), amounting to over 80% in the real estate case.

Linking. The runtime of the link discovery depends largely on the amount of data to link. In our use case, fetching all data items for linking from the endpoints required less than 3 minutes while the link discovery process itself was carried out in 0.6 seconds for discovering the near-by entities and 0.3 seconds for the entities at walking distance.

In summary, the data extraction and linking can be easily done in a few minutes per agency and can be run in parallel for multiple agencies. This allows us to refresh the data at least once per day, without overtaxing the resources of the agencies.

23.5 RELATED WORK

DEQA is, to the best of our knowledge, the first *comprehensive deep web question answering* system addressing the whole process from data extraction to question answering. In contrast, previous approaches have been limited either with respect to their access to deep web data behind scripted forms (Mollá and González, 2007) by targeting only common-sense, surface web data, or by requiring user action for form navigation (MORPHEUS, (Grant et al., 2010)). Though "federated" approaches that integrate data from different forms have been considered (Lin, 2002), none has integrated the extracted data with existing background knowledge, limiting the types of questions that can be answered. In the following, we briefly discuss related work for each of DEQA's components to illustrate why we believe this is the right combination.

Web Extraction. To extract the relevant data from the real estate agencies, we can resort essentially to three alternatives in web data information extraction (Chang et al., 2006), namely traditional information extraction, unsupervised data extraction, or supervised data extraction, with OxPath falling into the last category. *Information extraction* systems, such as (Etzioni et al., 2005, 2008), focus on extraction from plain text which is not suitable for deep web data extraction of product offers, where most of the data is published with rich visual and HTML structure, yielding much higher accuracy than IE systems. *Unsupervised data extraction* (Zhai and Liu, 2006; Kayed and Chang, 2010) approaches can use that structure, but remain limited in accuracy mostly due to their inability to distinguish relevant data from noise reliably. Thus, the only choice is a supervised approach. In (Furche et al., 2013) OxPath and related supervised approaches are discussed at length. In summary, OxPath presents a novel trade-off as a simpler, easier language with extremely high scalability at the cost of more sophisticated data analysis or processing capabilities. As shown in DEQA, such abilities are better suited for post-processing (e.g., through LIMES for linking).

Linking. The wealth of knowledge bases available in the Linked Data Cloud makes Link Discovery intrinsically complex w.r.t. its runtime. To address this issue, many time-efficient frameworks have been developed. LIMES (Ngonga Ngomo et al., 2011) offers a complex grammar for link specifications, and relies on a hybrid approach for computing complex link specifications. SILK (Isele et al., 2011) aims to place all instances that are to be compared in a multi-dimensional space, using MultiBlock to discard unnecessary comparisons efficiently. In contrast to LIMES, which employs lossless approaches, (Song and Heflin, 2011) uses a candidate selec-

tion approach based on discriminative properties to compute links very efficiently but potentially loses links while doing so. Other frameworks and approaches include those described in (Raimond et al., 2008; Glaser et al., 2009; Scharffe et al., 2009).

Albeit Link Discovery is closely related with record linkage and deduplication (Bleiholder and Naumann, 2008), it is important to notice that Link Discovery goes beyond these two tasks as Link Discovery aims to provide the means to link entities via arbitrary relations. Here, the database community has developed different blocking techniques to address the complexity of brute force comparison (Köpcke et al., 2009) and very time-efficient techniques to compute string similarities for record linkage (see e.g., (Xiao et al., 2008)). such as standard blocking, sorted-neighborhood, bi-gram indexing, canopy clustering and adaptive blocking have been developed by the database community to address the problem of the quadratic time complexity of brute force comparison (Köpcke et al., 2009). In addition, very time-efficient techniques have been proposed to compute string similarities for record linkage (Bayardo et al., 2007; Xiao et al., 2008) including All-Pairs (Bayardo et al., 2007), PPJoin and PPJoin+ (Xiao et al., 2008). However, these approaches alone cannot deal with the diversity of property values found on the deep web as, e.g., they cannot deal with numeric values, or can only deal with simple link specifications. In addition, most time-efficient string matching algorithms can only deal with simple link specifications, which are mostly insufficient when computing links between large knowledge bases.

In recent work, machine learning approaches have been proposed to discover link specifications. For example, (Song and Heflin, 2011) detect discriminative properties by using string concatenations, whereas RAVEN (Ngonga Ngomo et al., 2011) combines stable marriage algorithms and a perceptron-based learning algorithm within the frame of active learning to compute boolean and linear classifiers. For example (Ngonga Ngomo and Lyko, 2012) combine genetic programming and active learning while (Nikolov et al., 2012) learns link specifications in an unsupervised manner.

Question Answering. There is a range of approaches to QA over structured data, for an overview see (Lopez et al., 2011). Here we discuss TBSL in contrast to two prominent systems to exemplify two opposite key aspects: *PowerAqua* (Lopez et al., 2009), a purely data-driven approach, and *Pythia* (Unger and Cimiano, 2011), which heavily relies on linguistic knowledge. TBSL specifically aims at combining the benefits of a deep linguistic analysis with the flexibility and scalability of approaches focusing on matching natural language questions to RDF triples. This contrasts with *PowerAqua* (Lopez et al., 2009), an open-domain QA system that uses no linguistic knowledge and thus fails on questions containing quantifiers and comparisons, such as the most and more than. *Pythia* (Unger and Cimiano, 2011), on the other hand, is a system that relies on a deep linguistic analysis, yet requires an extensive, manually created domain-specific lexicon.

23.6 CONCLUSION

DEQA is a comprehensive framework for *deep web question answering*, which improves existing search functionality by combining web extraction, data integration and enrichment as well as question answering. We argue that recent advances allow the successful implementation of the DEQA framework and consider this to be

one of the prime examples for benefits of semantic web and artificial intelligence methods. We instantiate DEQA for the real estate domain in Oxford and show in an evaluation on 100 user queries that DEQA is able to answer a significant percentage correctly. In addition, we provided a cost analysis which describes the setup and maintenance effort for implementing DEQA in a particular domain. All used software components as well as the actual queries and used configuration files are freely available (http://aksw.org/projects/DEQA).

24 REAL-TIME OPEN INFORMATION EXTRACTION

PREAMBLE

The second area of application of LIMES considered in this thesis is open knowledge extraction. Here, the time-efficient algorithms provided by our framework allow for rapidly detecting similar property labels for clustering. The content of this chapter was published in (Gerber et al., 2013). The author implemented parts of the framework, co-wrote the paper and supervised the work presented herein.

24.1 INTRODUCTION

Implementing the original vision behind the Semantic Web requires the provision of a Web of Data which delivers timely data at all times. The foundational example presented in Berners-Lee et al's seminal paper on the Semantic Web (Berners-Lee et al., 2001) describes a software agent who is tasked to find medical doctors with a rating of excellent or very good within 20 miles of a given location at a given point in time. This requires having timely information on which doctors can be found within 20 miles of a particular location at a given time as well as having explicit data on the rating of said medical doctors. Even stronger timeliness requirements apply in decision support, where software agents help humans to decide on critical issues such as whether to buy stock or not or even how to plan their drive through urban centers. Furthermore, knowledge bases in the Linked Open Data (LOD) cloud would be unable to answer queries such as "Give me all news of the last week from the New York Times pertaining to the director of a company". Although the current LOD cloud has tremendously grown over the last years (Auer et al., 2013a), it delivers mostly encyclopedic information (such as albums, places, kings, etc.) and fails to provide up-to-date information that would allow addressing the information needs described in the examples above.

The idea which underlies our work is thus to alleviate this current drawback of the Web of Data by developing an approach that allows extracting RDF from unstructured (i.e., textual) data streams in a fashion similar to the live versions of the DBpedia[1] and LinkedGeoData[2] datasets. The main difference is yet that instead of relying exclusively on structured data like LinkedGeoData or on semi-structured data like DBpedia, we rely mostly on unstructured, textual data to generate RDF. By these means, we are able to unlock some of the potential of the document Web, of which up to 85% is unstructured (Gaag et al., 2009). To achieve this goal, our approach, dubbed RdfLiveNews, assumes that it is given unstructured data streams as input. These are deduplicated and then used as basis to extract patterns for relations between known resources. The patterns are then clustered to labeled relations which are finally used as basis for generating RDF triples. We evaluate our approach against a sample of the RDF triples we extracted from RSS feeds and show that we achieve a very high precision.

1 http://live.dbpedia.org/sparql
2 http://live.linkedgeodata.org/sparql

The remainder of this work is structured as follows: We first give an overview of our approach and give detailed insights in the different steps from unstructured data streams to RDF. Then, we evaluate our approach in several settings. We then contrast our approach with the state of the art and finally conclude.

24.2 OVERVIEW

We implemented the general architecture of our approach dubbed RdfLiveNews according to the pipeline depicted in Figure 24.1. First, we gather textual data from data streams by using RSS feeds of news articles. Our approach can yet be employed on any unstructured data published by a stream. Since input streams from the Web can be highly redundant (i.e., convey the same information), we then deduplicate the set of streams gathered by our approach. Subsequently, we apply a pattern search to find lexical patterns for relations expressed in the text. After a refinement step with background knowledge, we finally cluster the extracted patterns according to their semantic similarity and transform this information into RDF.

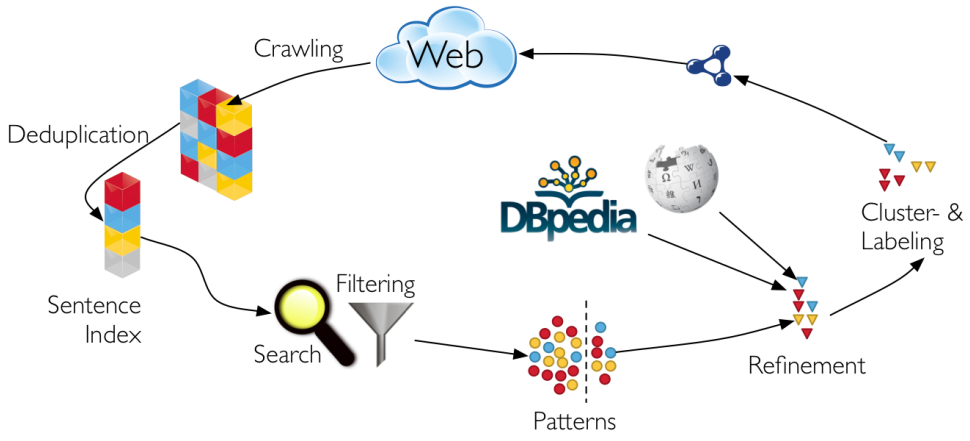


Figure 24.1: Overview of the generic time slice-based stream processing

24.2.1 *Data Acquisition*

Formally, our approach aims to process the output of unstructured data sources S^i by continuously gathering the data streams D^i that they generate. Each data stream consists of atomic elements d^i_j (in our case sentences). Let $D^i_{[t,t+d]}$ be the portion of D^i that was emitted by S^i between the times t and $t+d$. The data gathering begins by iteratively gathering the elements of the streams $D^i_{[t,t+d]}$. from all available sources S^i for a period of time d, which we call the *time slice duration*. For example, this could mean crawling a set of RSS feeds for a duration of 2 hours. We call $D^i_{[t,t+d]}$ a slice of D^i. We will assume that we begin this process at $t = 0$, thus leading to slices $D^i_{[k.d,(k+1).d]}$ with $k \in \mathbb{N}$. The data gathered from all sources during a time slice duration is called a *time slice*. We apply sentence splitting on all slices to generated their elements.

24.2.2 *Deduplication*

The aim of the deduplication step is to remove very similar elements from slices before the RDF extraction. This removal accounts for some Web data streams simply repeating the content of one of several other streams. Our deduplication approach is based on measuring the similarity of single elements s_i and s_j found in unstructured streams. Elements of streams are considered to be different iff $qgrams(s_i, s_j) < \theta$, where $\theta \in [0, 1]$ is a similarity threshold and $qgrams(s_i, s_j)$ measures the similarity of two strings by computing the Jaccard similarity of the trigrams they contain. Given that the number of stream items to deduplicate can be very large, we implemented the following two-step approach: For each slice $D^i_{[k.d,(k+1)d]}$, we first deduplicate the elements s^i_j within $D^i_{[k.d,(k+1)d]}$. This results in the following duplicate-free data stream $\Delta^i_{[k.d,(k+1)d]}$ whose elements d^i_j fulfill the following equations:

$$d^i_j \in D^i_{[k.d,(k+1)d]}, \tag{24.1}$$

$$\forall s^i_k \in D^i_{[k.d,(k+1)d]}\ \exists d^i_j \in \Delta^i_{[k.d,(k+1)d]}\ qgrams(s^i_k, d^i_j) \geqslant \theta, \tag{24.2}$$

$$\forall d^i_j, d^i_k \in \Delta^i_{[k.d,(k+1)d]}\ qgrams(d^i_k, d^i_j) < \theta). \tag{24.3}$$

The elements of $\Delta^i_{[k.d,(k+1)d]}$ are then compared to all other elements of the w previous deduplicated streams $\Delta^i_{[(k-1).d,kd]}$ to $\Delta^i_{[(k-w).d,(k-w+1)d]}$, where w is the size of the deduplication window. Only $\Delta^i_{[k.d,(k+1)d]}$ is used for further processing. To ensure the scalability of the deduplication step, we are using deduplication algorithms implemented in the LIMES framework (Ngonga Ngomo and Auer, 2011). Table 24.2 gives an overview of the number of unique data stream items in our dataset when using different deduplication thresholds.

24.2.3 *Pattern Search and Filtering*

In order to find patterns we first apply Entity Recognition (NER) and Part of Speech (POS) tagging on the deduplicated sentences. RdfLiveNews can use two different ways to extract patterns from annotated text. The POS tag method uses *NNP* and *NNPS*[3] tagged tokens to identify a relation's subject and object, whereas the Entity Tag method relies on *Person*, *Location*, *Organization* and *Miscellaneous* tagged tokens. In an intermediate step all consecutive POS and NER tags are merged. An unrefined RdfLiveNews pattern p is now defined as a pair $p = (\theta, \mathcal{S}_\subseteq)$, where θ is the natural language representation (NLR) of p and $\mathcal{S}_\subseteq = \{(s_i, o_i) : i \in \mathbb{N}; 1 \leqslant i \leqslant n\}$ is the support set of θ, a set of the subject and object pairs. For example the sentence:

David/NNP hired/VBD John/NNP ,/, former/JJ manager/NN of/IN ABC/NNP ./.

would result in the patterns:

$p_1 = (\ [hired], \{(David,\ John)\}$ and

3 All POS tags can be found in the Penn Treebank Tagset.

$p_2 = ([,\ \text{former manager of}],\{(\text{John},\ \text{ABC})\})$.

After the initial pattern acquisition step, we filter all patterns to improve their quality. We discarded all patterns that did not match these criteria: The pattern should (1) contain at least a verb or a noun, (2) contain at least one salient word (i.e. a word that is not a stop word), (3) not contain more than one non-alpha-numerical character (except ", ' '") and (4) be shorter than 50 characters. Since the resulting list still contains patterns of low quality, we first sort it by the number of elements of the support set $\mathcal{S}_\theta$ and solely select the top 1% for pattern refinement to ensure high quality.

24.2.4 *Pattern Refinement*

The goal of this step is to find a suitable *rdfs:range* and *rdfs:domain* as well as to disambiguate the support set of a given pattern. To achieve this goal we first try to find an URI for the subjects and objects in the support set of p by matching the pairs to entries in a knowledge base. With the help of those URIs we can query the knowledge base for the classes (*rdf:type*) of the given resources and compute a common *rdfs:domain* for the subjects of p and *rdfs:range* for the objects respectively. A refined RdfLiveNews pattern p_r is now defined as a quadruple $p_r = (\theta, \mathcal{S}_\subseteq{}', \delta, \rho)$, where θ is the natural language representation, $\mathcal{S}_\subseteq{}'$ the disambiguated support set, δ the *rdfs:domain* and ρ the *rdfs:range* of p_r.

To find the URIs of each subject-object pair $(s, o) \in \mathcal{S}_\subseteq$ we first try to complete the entity name. This step is necessary and beneficial because entities usually get only written once in full per article. For example the newly elected president of the United States of America might be referenced as "President Barack Obama" in the first sentence of a news entry and subsequently be referred to as "Obama". In order to find the subjects' or objects' full name, we first select all entities $e \in \mathcal{E}_a$ of the article the pair (s, o) was found in. We then use the longest matching substring between s (or o) and all elements of $\mathcal{E}_a$ as the name of s or o respectively. Additionally we can filter the elements of $\mathcal{E}_a$ to contain only certain NER types. Once the complete names of the entities are found, we can use them to generate a list of URI candidates $\mathcal{C}_{uri}$. This list is generated with the help of a query for the given entity name on a list of surface forms (e.g. "U.S." or "USA" for the *United States of America*), which was compiled by analyzing the *redirect* and *disambiguation* links from Wikipedia as presented in (Mendes et al., 2011). Each URI candidate $c \in \mathcal{C}_{uri}$ is now evaluated on four different features and the combined score of those features is used to rank the candidates and choose the most probable URI for an entity. The first feature is the *Apriori*-score $a(c)$ of the URI candidate c, which is calculated beforehand for all URIs in the knowledge base by analyzing the number of inbound links of c by the following formula: $a(c) = \log(\text{inbound}(c) + 1)$. The second and third features are based on the context information found in the Wikipedia article of c and the news article text (s, o) was found in. For the *global context*-score c_g we apply a co-occurrence analysis of the entities $\mathcal{E}_a$ found in the news article and the entities $\mathcal{E}_w$ found in the Wikipedia article of c. The *global context*-score is now computed as $c_g(\mathcal{E}_a, \mathcal{E}_w) = |\mathcal{E}_a \cap \mathcal{E}_w| \,/\, |\mathcal{E}_a \cup \mathcal{E}_w|$. The *local context*-score c_l is the number of mentions of the second element of the pair (s, o), o in the case of s and vice versa, in $\mathcal{E}_w$. The last feature to determine a URI for an entity is the maximum string similarity sts between s (or o) and the

elements of the list of surface forms of c. We used the qgram distance[4] as the string similarity metric. We normalize all non-$[0,1]$ features (c_g, c_l, a) by applying a minimum-maximum normalization of the corresponding scores for $\mathcal{C}_{uri}$ and multiply it with a weight parameter which leads to the overall URI score:

$$c(s,o,uri) = \frac{\frac{\alpha a}{a_{max}} + \frac{\beta c_g}{c_{g_{max}}} + \frac{\gamma c_l}{c_{l_{max}}} + \delta sts}{4}$$

If the URI's score is above a certain threshold $\lambda \in [0,1]$ we use it as the URI for s, otherwise we create a new URI. Once we have computed the URIs for all pairs $(s,o) \in \mathcal{S}_{\subseteq}$ we determine the most likely *domain* and *range* for p_r. This is done by analyzing the *rdf:type* statements returned for each subject or object in $\mathcal{S}_{\subseteq}$ from a background knowledge base. Since the DBpedia ontology is designed in such a way, that classes do only have one super-class, we can easily analyze its hierarchy. We implemented two different determination strategies for analyzing the class hierarchy. The first strategy, dubbed "most general", selects the highest class in the hierarchy for each subject (or object) and uses the most occurring class as *domain* or *range* of p_r. The second strategy, dubbed "most specific", works similar to the "most general" strategy with the difference that it uses the most descriptive class to select the *domain* and *range* of p_r.

24.2.5 *Pattern Similarity and Clustering*

In order to cluster patterns according to their meaning, we created a set of similarity measures. A similarity measure takes two patterns p_1 and p_2 as input and outputs the similarity value $s(p_1,p_2) \in [0,1]$. As a baseline we implemented a qgram measure, which calculates the string similarity between all non stop words of two patterns. Since this baseline measure fails to return a high similarity for semantically related, but not textually similar patterns like "'s attorney ," and "'s lawyer ," we also implemented a Wordnet measure. As a first step the Wordnet similarity measure filters out the stop words of p_1 and p_2 and applies the Stanford lemmatizer on the remaining tokens. Subsequently, for all token combinations of p_1 and p_2, we apply a Wordnet Similarity metric (Path (Pedersen et al., 2004), Lin (Lin, 1998) and Wu & Palmer (Wu and Palmer, 1994)) and select the maximum of all comparisons as the similarity value $s(p_1,p_2)$. As a final similarity measure we created a Wordnet and string similarity measure with the help of a linear combination from the before-mentioned metrics. In this step we also utilize the *domain* and *range* of p_r. If this feature is enabled, a similarity value between two patterns p_1 and p_2 can only be above 0, iff $\{\delta_{p_1}, \rho_{p_1}\} \setminus \{\delta_{p_2}, \rho_{p_2}\} = \emptyset$.

The result of the similarity computation can be regarded as a similarity graph $G = (V, E, \omega)$, where the vertices are patterns and the weight $\omega(p_1,p_2)$ of the edge between two patterns is the similarity of these patterns. Consequently, unsupervised machine learning and in particular graph clustering is a viable way of finding groups of patterns that convey similar meaning. We opted for using the BorderFlow clustering algorithm (Ngonga Ngomo and Schumacher, 2009) as it is parameter-free and has already been used successfully in diverse applications including clustering protein-protein interaction data and queries for SPARQL benchmark creation (Morsey et al., 2011, 2012). For each node $v \in V$, the algorithm

4 http://sourceforge.net/projects/simmetrics/

begins with an initial cluster X containing only v. Then, it expands X iteratively by adding nodes from the direct neighborhood of X to X until X is node-maximal with respect to the border flow ratio described in (Ngonga Ngomo and Schumacher, 2009; Morsey et al., 2011). The same procedure is repeated over all nodes. As different nodes can lead to the same cluster, identical clusters (i.e., clusters containing exactly the same nodes) that resulted from different nodes are subsequently collapsed to one cluster. The set of collapsed clusters and the mapping between each cluster and the nodes that led to it are returned as result.

24.2.6 *Cluster Labeling and Merging*

Based on the clusters $\mathcal{C}$ obtained through the clustering algorithm, this step selects descriptive labels for each cluster $c_i \in \mathcal{C}$, which can afterwards be used to merge the clusters. In the current version, we apply a straightforward majority voting algorithm, i.e. for each cluster c_i, we select the most frequent natural language representation θ (stop words removed) occurring in the patterns of c_i. Finally, we use the representative label of the clusters to merge them using a string similarity and WordNet based similarity measure. This merging procedure can be applied repeatedly to further reduce the number of clusters, but taking into account that those similarity measures are not transitive, we are currently only running it once, as we're more focused on accuracy.

24.2.7 *Mapping to RDF and Publication on the Web of Data*

To close the circle of the round-trip pipeline of RdfLiveNews, the following prerequisite steps are required to re-publish the extraction results in a sensible way:

1. The facts and properties contained in the internal data structure of our tool have to be mapped to OWL.
2. Besides the extracted factual information several other aspects and meta data are interesting as well, such as extraction and publication data and provenance links to the text the facts were extracted from.
3. URIs need to be minted to provide the extracted triples as Linked Data.

Mapping to OWL. Each cluster $c_i \in \mathcal{C}$ represents an *owl:ObjectProperty* $\mathtt{prop}_{c_i}$. The *rdfs:domain* and *rdfs:range* of $\mathtt{prop}_{c_i}$ is determined by a majority voting algorithm with respect to δ and ρ of all $p_r \in \mathcal{C}$. The *skos:prefLabel*[5] of $\mathtt{prop}_{c_i}$ is the label determined by the cluster labeling step and all other NLRs of the patterns in c_i get associated with $\mathtt{prop}_{c_i}$ as *skos:altLabels*. For each subject-object pair in $\mathcal{S}_{\subseteq}{}'$ we produce a triple by using $\mathtt{prop}_{c_i}$ as predicate and by assigning learned entity types from DBpedia or *owl:Thing*.

Provenance tracking with NIF. Besides converting the extracted facts from the text, we are using the current draft of the NLP Interchange Format (NIF) Core ontology[6] to serialize the following information in RDF: the sentence the triple was extracted from, the extraction date of the triple, the link to the source URL of the data stream item and the publication date of the item on the stream. Furthermore,

5 http://www.w3.org/2004/02/skos/
6 http://persistence.uni-leipzig.org/nlp2rdf/ontologies/nif-core#

NIF allows us to link each element of the extracted triple to its origin in the text for further reference and querying.

NIF is an RDF/OWL based format to achieve interoperability between language tools, annotation and resources. NIF offers several URI schemes to create URIs for strings, which can then be used as subjects for annotation. We employ the NIF URI scheme, which is grounded on URI fragment identifiers for text (RFC 5147[7]). For our use case, we extended NIF in two ways: (1) we added the ability to represent extracted triples via the ITS 2.0 / RDF Ontology[8]. *itsrdf:taPropRef* is an *owl:AnnotationProperty* that links the NIF String URI to the *owl:ObjectProperty* by RdfLiveNews. The three links from the NIF String URIs (str_1, str_2, str_3) to the extracted triple (s, p, o) itself make it well traceable and queryable: $str_1 \mapsto s$, $str_2 \mapsto p$, $str_3 \mapsto o$, $s \mapsto p \mapsto o$. An example of NIF RDF serialization is shown in Listing 24.1. (2) Although (Rizzo et al., 2012) already suggested the minting of new URIs, a concrete method for doing so was not yet researched. In RdfLiveNews we use the source URL of the data stream item to re-publish the facts for individual sentences as Linked Data. We strip the scheme component (http://) of the source URL and percent encode the ultimate part of the path and the query component[9] and add the md5 encoded sentence to produce the following URI:

```
http://rdflivenews.aksw.org/extraction/ + example.com:8042/over/ +
    urlencode(there?name=ferret) + / + md5(‘sentence‘)
```

Listing 24.1: Example RDF extraction of RdfLiveNews

```
@base <http://rdflivenews.aksw.org/extraction/www.necn.com/07/04/12/Scientists
    -discover-new-subatomic-partic/landing.html%3FblockID%3D735470%26feedID%3
    D4213/8a1e5928f6815c99b9d2ce613cf24198#>.
## prefixes: please use http://prefix.cc, e.g. http://prefix.cc/rlno
## extracted property + result of linking
rlno:directorOf a owl:ObjectProperty ;
  skos:prefLabel "director of" , skos:altLabel ", director of " ;
  owl:equivalentProperty dbp:director .
## extracted facts:
rlnr:Rolf_Heuer a dbo:Person ;
  rdfs:label "Rolf Heuer"@en ;
  rlno:directorOf dbpedia:CERN .
dbpedia:CERN a owl:Thing ;
  rdfs:label "CERN"@en .
## provenance tracking with NIF:
<char=0,10>   itsrdf:taClassRef dbo:Person ;
  itsrdf:taIdentRef rlnr:Rolf_Heuer .
<char=14,18>  itsrdf:taIdentRef dbpedia:CERN .
<char=11,24>  nif:anchorOf      ", director of"^^xsd:string ;
  itsrdf:taPropRef rlno:directorOf .
## detailed NIF output with context, indices and anchorOf
<char=0,> a nif:String, nif:Context, nif:RFC5147String ;
  nif:isString "Rolf Heuer , director of CERN , said the newly discovered
      particle is a boson , but he stopped just shy of claiming outright that
      it is the Higgs boson itself - an extremely fine distinction." ;
  nif:sourceUrl <http://www.necn.com/07/04/12/Scientists-discover-new-
      subatomic-partic/landing.html?blockID=735470&feedID=4213>;
## extraction date:
```

7 http://tools.ietf.org/html/rfc5147
8 http://www.w3.org/2005/11/its/rdf#
9 http://tools.ietf.org/html/rfc3986#section-3

```
  dcterms:created "2013-05-09T18:27:08+02:00"^^xsd:dateTime .
## publishing date:
<http://www.necn.com/07/04/12/Scientists-discover-new-subatomic-partic/landing
    .html?blockID=735470&feedID=4213>
  dcterms:created "2012-08-15T14:48:47+02:00"^^xsd:dateTime .
<char=0,10> a nif:String, nif:RFC5147String ;
  nif:referenceContext <char=0,>; nif:anchorOf "Rolf Heuer" ;
  nif:beginIndex "0"^^xsd:long ; nif:endIndex "10"^^xsd:long ;
```

Republication of RDF. The extracted triples are hosted on: http://rdflivenews.aksw.org. The data for individual sentences is crawlable via the file system of the Apache2 web server. We assume that source URLs only occur once in a stream when the document is published and the files will not be overwritten. Furthermore, the extracted properties and entities are available as Linked Data. The template for the URI generation is *http://rdflivenews.aksw.org/{ontology | resource}/$name*. In addition, the extraction results can be queried via SPARQL at http://rdflivenews.aksw.org/sparql.

24.2.8 *Linking*

The approach described above generates a set of properties with several labels. In our effort to integrate this data source into the Linked Open Data Cloud, we use the deduplication approach proposed in Section 24.2.2 to link our set of properties to existing knowledge bases (e.g., DBpedia). To achieve this goal, we consider the set of properties we generated as set of source instances S while the properties of the knowledge base to which we link are considered to be a set of target T. Two properties $s \in S$ and $t \in T$ are linked iff $\mathtt{trigrams}(s,t) \geqslant \theta_p$, where $\theta_p \in [0,1]$ is the property similarity threshold.

24.3 EVALUATION

The aim of our evaluation was to answer four questions. First, we aimed at testing how well RdfLiveNews is able to disambiguate found entities. Our second goal was to determine if the proposed similarity measures can be used to cluster patterns with respect to their semantic similarity. Third, we wanted to evaluate the quality of the RDF extraction and linking. Finally, we wanted to measure if all computational heavy tasks can be applied in real-time, meaning the processing of one iteration takes less time than its compilation.

For this evaluation we used a list of 1457 RSS feeds as compiled in (Goldhahn et al., 2012). This list includes all major worldwide newspapers and a wide range of topics, e.g. *World*, *U.S.*, *Business*, *Science* etc. We crawled this list for 76 hours, which resulted in a corpus, dubbed *100%* of 38 time slices of 2 hours and 11.7 million sentences. The average number of sentences per feed entry is approximately 26.5 and there are 3445 articles on average per time slice. Additionally we created two subsets of this corpus by randomly selecting 1% and 10% of the contained sentences. All evaluations were carried out on a MacBook Pro with a quad-core Intel Core i7 (2GHz), a solid state drive and 16 GB of RAM.

24.3.1 *URI Disambiguation*

To evaluate the URI disambiguation we created a gold standard manually. We took the 1% corpus, applied deduplication with a window size of 40 (contains all time slices) and a threshold of 1 (identical sentences), which resulted in a set of 69884 unique sentences. On those sentences we performed the pattern extraction with part of speech tagging as well as filtering. In total we found 16886 patterns and selected the Top 1%, which have been found by 1729 entity pairs. For 473 of those entity pairs we manually selected a URI for subject and object. This resulted in an almost equally distributed gold standard with 456 DBpedia and 478 RdfLiveNews URIs. We implemented a hill climbing approach with random initialization to optimize the parameters (see Section 24.2.4). The precision of our approach is the ratio between correctly found URIs for subject and object to the number of URIs above the threshold λ as shown in Equation 24.4. The recall, shown in Equation 24.5, is determined by the ratio between the number of correct subject and object URIs and the total number of subjects and objects in the gold standard. The F_1 measure is determined as usual by: $F_1 = 2 \cdot \frac{P \cdot R}{P+R}$. We optimized our approach for precision since we can compensate a lower recall and could achieve a precision of **85.01%** where the recall is **40.69%** and the resulting F_1 is **55.03%**. The parameters obtained through the hill-climbing search indicate that the *Apriori*-score is the most influential parameter (1.0), followed by *string-similarity* (0.78), *local-context* (0.6), global context (0.45) and a URI score threshold of 0.61. If we optimize for F_1, we were able to achieve a F_1 measure of **66.49%** with a precision of **67.03%** and a recall of **65.95%**.

For 487 out of the 934 URI in the gold standard no confident enough URI could be found. The most problems occured for DBpedia URIs which could not be determined in 305 cases, in comparison to 182 URIs for newly created resources. Additionally, for 30 resources RdfLiveNews created new URIs where DBpedia URIs should be used and in 0 cases a DBpedia URI was used where a new resource should be created. The reason for those mistakes are tagging errors, erroneous spellings and missing context information. For example Wikipedia has 97 disambiguations for "John Smith" which can not be disambiguated without prior knowledge.

We used AIDA (Yosef et al., 2011) to compare our results with a state-of-the-art NED algorithm. We configured AIDA with the Cocktailparty setup, which defines the recommended configuration options of AIDA. AIDA achieved an accuracy of 0.57, i.e. 57% of the identifiable entities were correctly disambiguated. The corpus described above provides a difficult challenge due to the small disambiguation contexts and is limited to graphs evolving from two entities per text. AIDA tries to build dense sub-graphs in a greedy manner in order to perform correct disambiguation. This algorithm would profit from a bigger number of entities per text. The drawback is AIDA needs 2 minutes to disambiguate 25 sentences. Overall, AIDA performs well on arbitrary entities.

$$P = \frac{|s_{uri_c}| + |o_{uri_c}|}{|s_{uri}| + |o_{uri}|} \qquad (24.4)$$

$$R = \frac{|s_{uri_c}| + |o_{uri_c}|}{2 \cdot |GS|} \qquad (24.5)$$

24.3.2 *Pattern Clustering*

To evaluate the similarity generation as well as the clustering algorithm we relied on the measures Sensitivity, Positive Predictive Value (PPV) and Accuracy. We used the adaptation of those measures as presented in (Brohée and van Helden, 2006) to measure the match between a set of pattern mappings[10] from the gold standard and a clustering result. The gold standard was created by clustering the patterns as presented in the previous section manually. This resulted in a list of 25 clusters with more than 1 pattern and 54 clusters with 1 pattern. Since cluster with a size of 1 would skew our evaluation into unjustified good results, we excluded them from this evaluation.

Sensitivity. With respect to the clustering gold standard, we define sensitivity as the fraction of patterns of pattern mapping i which are found in cluster j. In $Sn_{i,j} = T_{i,j}/N_i$, N_i is the number of patterns belonging to pattern mapping i. We also calculate a pattern mapping-wise sensitivity Sn_{pm_i} as the maximal fraction of patterns of pattern mapping i assigned to the same cluster. $Sn_{pm_i} = max_{j=1}^{m} Sn_{i,j}$ reflects the coverage of pattern mapping i by its best-matching cluster. To characterize the general sensitivity of a clustering result, we compute a clustering-wise sensitivity as the weighted average of Sn_{pm_i} over all pattern mappings: $Sn = \frac{\sum_{i=1}^{n} N_i Sn_{pm_i}}{\sum_{i=1}^{n} N_i}$.

Positive Predictive Value. The positive predictive value is the proportion of members of cluster j which belong to pattern mapping i, relative to the total number of members of this cluster assigned to all pattern mappings. $PPV_{i,j} = T_{i,j}/\sum_{i=1}^{n} T_{i,j} = T_{i,j}/T_{.j}$
$T_{.j}$ is the sum of column j. We also calculate a cluster-wise positive predictive value PPV_{cl_j}, which represents the maximal fraction of patterns of cluster j found in the same annotated pattern mapping. $PPV_{cl_j} = max_{i=1}^{n} PPV_{i,j}$ reflects the reliability with which cluster j predicts that a pattern belongs to its best-matching pattern mapping. To characterize the general PPV of a clustering result as a whole, we compute a clustering-wise PPV as the weighted average of PPV_{cl_j} over all clusters: $PPV = \frac{\sum_{j=1}^{m} T_{.j} PPV_{cl_j}}{\sum_{j=1}^{m} T_{.j}}$.

Accuracy. The geometric accuracy (Acc) indicates the tradeoff between sensitivity and positive predictive value. It is obtained by computing the geometrical mean of the Sn and the PPV: $Acc = \sqrt{Sn \cdot PPV}$.

We evaluated the three similarity measures with respect to the underlying WordNet similarity metric (see Section 24.2.5). Furthermore we varied the clustering similarity threshold between 0.1 and 1 with a 0.1 step size. In case of the qgram and WordNet similarity metric we performed a grid search on the WordNet and qgram parameter in $[0, 1]$ with a step size of 0.05. We achieved the best configuration with the qgram and WordNet similarity metric with an accuracy of **82.45%**, a sensitivity of **71.17%** and a positive predictive value of **95.51%**. The best WordNet metric is Lin, the clustering threshold 0.3 and the qgram parameter is with 0.45 significantly less influential than the WordNet parameter with 0.75. As a reference value, the plain WordNet similarity metric achieved an accuracy of 78.86% and the qgram similarity metric an accuracy of 69.1% in their best configuration.

10 A pattern mapping maps NLRs to RDF properties.

24.3.3 *RDF Extraction and Linking*

To assess the quality of the RDF data extracted by RdfLiveNews, we sampled the output of our approach and evaluated it manually. We generated five different evaluation sets. Each set may only contain triples with properties of clusters having at least $i = 1 \ldots 5$ patterns. We selected 100 triples (if available) randomly for each test set. As the results in Table 24.1 show, we achieve high accuracy on subject and object disambiguation. As expected, the precision of our approach grows with the threshold for the minimal size of clusters. This is simply due to the smaller clusters having a higher probability of containing outliers and thus noise.

E_i	1	2	3	4	5
S_{Acc}	0.81	0.88	0.86	0.857	0.804
P_{Acc}	0.86	0.89	0.90	0.935	1.00
O_{Acc}	0.93	0.91	0.90	0.948	0.941
$Total_{Acc}$	0.86	0.892	0.885	0.911	0.906
$\lvert E_i \rvert$	100	100	100	77	51
$\lvert P \rvert \in \lvert E_i \rvert$	28	22	12	6	1

Table 24.1: Accuracy of RDF Extraction for subject (S), predicates (P) and objects (O) on 1% dataset with varying cluster sizes E_i

The results of the linking with DBpedia (see Table 24.3) showed the mismatch between the relations that occur in news and the relations designed to model encyclopedic knowledge. While some relations such as `dbo:director` are used commonly in news streams and in the Linked Data Cloud, relations with a more volatile character such as `rlno:attorney` which appear frequently in news text are not mentioned in DBpedia.

Time Slice	No deduplication	$\theta = 1.0$	$\theta = 0.95$	$\theta = 0.9$
1	2997	2764	2764	2759
5	3047	2335	2334	2327
10	3113	2033	2040	2022
15	2927	1873	1868	1866
20	3134	1967	1966	1949
25	3065	1936	1932	1924
30	3046	1941	1940	1933

Table 24.2: Number of non-duplicate sentences in 1% of the data extracted from 1457 RSS feeds within a window of 10 time slices (2h each). The second column shows the original number of sentences without duplicate removal.

RdfLiveNews-URI	DBpedia-URI	Sample of cluster
rlno:directorOf	dbo:director	[manager], [, director of], [, the director of]
rlno:spokesperson	dbo:spokesperson	[, a spokeswoman for], [spokesperson], [, a spokesman for]
rlno:attorney	—	['s attorney ,], ['s lawyer ,], [attorney]

Table 24.3: Example for linking between RdfLiveNews and DBpedia

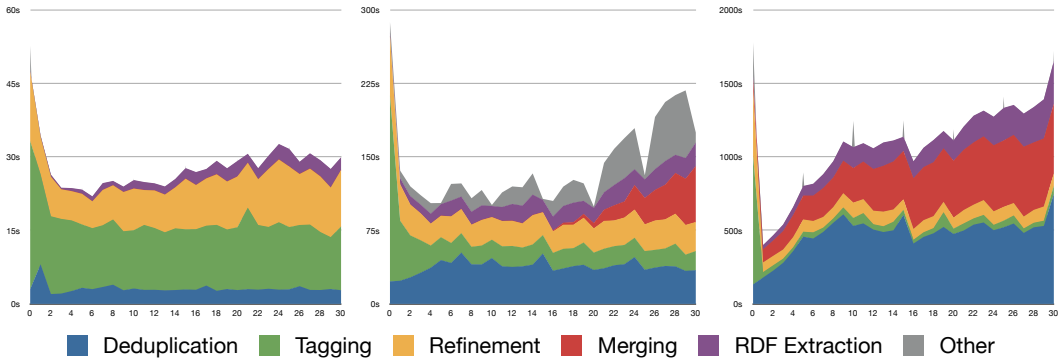


Figure 24.2: Runtimes for different components and corpora (1% left, 10% middle, 100% right) per iteration

24.3.4 *Scalability*

In order to perform real-time RDF extraction, the processing of the proposed pipeline needs to be done in less time than its acquisition requires. This also needs to be true for a growing list of RSS feeds. Therefore, we analyzed the time each module needed in each iteration and compared these values between the three test corpora. An early approximation of this evaluation implied that the pipeline indeed was not fast enough, which led to the parallelization of the pattern refinement and similarity generation. The results of this evaluation can be seen in Figure 24.2. With an average time slice processing time of about 20 minutes for the 100% corpus (2.2 minutes for 10% and 30s for 1%), our approach is clearly fit to handle up to 1500 RSS and more. The spike in the first iteration results out of the fact that RSS feeds contain the last n previous entries, which leads to a disproportional large first time slice. The most time consuming modules are the deduplication, tagging and cluster merging. To tackle these bottlenecks we can for example parallelize sentence tagging and the deduplication.

The results of the growth evaluation for patterns until iteration 30 can be seen in Figure 24.3. The number of patterns grows with the factor of 3 from 1% to 10% and 10% to 100% corpora. Also, the number of patterns found by more than one subject-object pair increases approximately by factor 2. Additionally we observed a linear growth for all patterns (also for patterns with $|\mathcal{S}'_\theta| > 1$) and 100% showing the highest growth rate with a factor 2.5 over 10% and 4.8 over 10%.

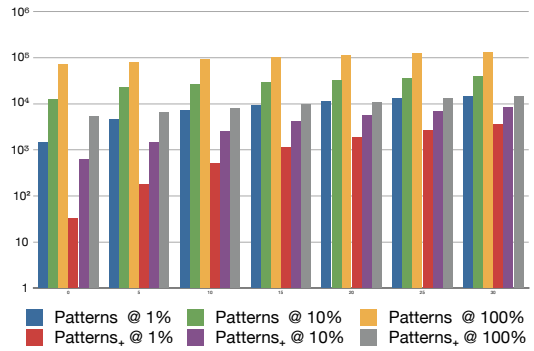


Figure 24.3: Number of patterns (log scale) and patterns with $|\mathcal{S}'_\theta| > 1$ (Patterns$_+$) for iterations and test corpus

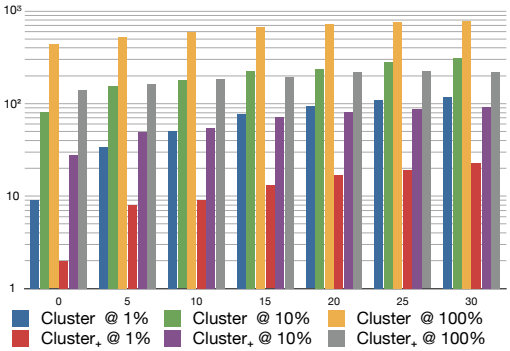


Figure 24.4: Number of clusters (log scale) and clusters with $|\mathcal{C}| > 1$ (Cluster$_+$) for iterations and test corpus

The results of the growth evaluation for clusters can be seen in Figure 24.4. The evaluation shows that the number of clusters increases by a factor of 2.5 from 1% to 10% and 10% to 100%. Moreover, approximately 25% of all cluster have more than 1 pattern and the number of clusters grows linear for 1% and 10% but for the 100% corpus it seems to coverage to 800. The same holds true for clusters with more then one pattern, as they stop to grow at around 225 clusters.

24.4 RELATED WORK

While Semantic Web applications rely on formal, machine understandable languages such as RDF and OWL, enabling powerful features such as reasoning and expressive querying, humans use Natural Language (NL) to express semantics. This gap between the two different languages has been filled by Information Extraction (IE) approaches, developed by the Natural Language Processing (NLP) research community (Sarawagi, 2008), whose goal is to find desired pieces of information, such as concepts (hierarchy of terms which are used to point to shared definitions), entities (numeric expression, date) and facts in natural language texts and print them in a form that is suitable for automatic querying and processing. Ever since the advent of the Linked Open Data initiative[11], IE is also an important key enabler for the Semantic Web. For example, LODifier (Augenstein et al., 2012) combines deep semantic analysis with entity recognition, word-sense disambiguation and controlled Semantic Web vocabularies. FOX (Ngonga Ngomo et al., 2011; Speck and Ngonga Ngomo, 2014) uses ensemble learning to improve the F-score of IE tools. The BOA framework (Gerber and Ngonga Ngomo, 2012) uses structured data as background knowledge for the extraction of natural language patterns, which are subsequently employed to extract additional RDF data from natural language text. The authors of (Nakashole and Weikum, 2012) propose a simple model for fact extraction in real-time taking into account the difficult challenges that timely fact extraction on frequently updated data entails. A specific application for the news domain is described in (Stern and Sagot, 2012), wherein a knowledge base of entities for the French news agency AFP is populated.

State-of-the-art open-IE systems such as ReVerb automatically identify and extract relationships from text, relying on (in the case of ReVerb) simple syntactic

11 http://linkeddata.org/

constraints expressed by verbs (Fader et al., 2011). The authors of (Davidov and Rappoport, 2008) present a novel pattern clusters method for nominal relationship classification using an unsupervised learning environment, which makes the system domain and language-independent. (Ruiz-Casado et al., 2007) shows how lexical patterns and semantic relationships can be learned from concepts in Wikipedia.

24.5 CONCLUSION AND FUTURE WORK

In this paper, we presented RdfLiveNews, a framework for the extraction of RDF from unstructured data streams. We presented the components of the RdfLiveNews framework and evaluated its disambiguation, clustering, linking and scalability capabilities as well as its extraction quality. We are able to disambiguate resources with a precision of 85%, cluster patterns with an accuracy of 82.5% and extract RDF with an total accuracy of around 90% and handle two hour time slices with around 300.000 sentences within 20 min on a small server. In future work, we will extend our approach to also cover datatype properties. For example from the sentence "..., Google said Motorola Mobility contributed revenue of US$ 1.25 billion for the second quarter." the triple *dbpedia:Google rlno:says "Motorola Mobility contributed revenue of US$ 1.25 billion for the second quarter"* can be extracted. Additionally we plan to integrate DeFacto (Lehmann et al., 2012), which is able to verify or falsify a triple extracted by RdfLiveNews. Finally, we will extend our approach with temporal logics to explicate the temporal scope of the triples included in our knowledge base.

25 LINKED CANCER GENOME ATLAS DATABASE

PREAMBLE

The last three chapters of this part present how LIMES was used for the purpose for which it was initially designed, i.e., for linking knowledge bases. In this chapter, we focus on LinkedTCGA,[1]. This dataset was derived from The Cancer Genome Atlas (TCGA), a multidisciplinary, multi-institutional pilot project to create an atlas of genetic mutations responsible for cancer. The dataset was linked with a subset of Bio2RDF via LIMES. The author co-supervised this work, implemented the linking to Bio2RDF and co-wrote the corresponding paper (Saleem et al., 2013), the content of which is presented below.

25.1 INTRODUCTION

The Cancer Genome Atlas (TCGA)[2] is an effort led by the National Cancer Institute[3] and aims to characterize and sequence 33 cancer types from 9000 patients at the molecular level. The ultimate goal of the project is to collect and make publicly available the data necessary to produce an Atlas of the genomic alterations responsible for the initiation and progression of cancer. TCGA offers data categorized into three data levels: raw data (level 1), normalized data (level 2) and processed data (level 3). To date, a total of 21 types of data have been collected for each patient, making up a total of 147,645 raw data files, of which 53,694 contain level 3 (processed) data, summing up to a total of 12.7 terabytes of data. According to information in the TCGA portal, this is only 46% of the expected data with new data being submitted every day. In this paper, only level 3 data is of interest as it is the data upon which analytics is performed.

TCGA is a valuable resource for hypothesis-driven translational research as all of its data results from direct experimental evidence. Analysis of such evidence within cancer research has led in recent years to clinically relevant findings in the genetic mark-ups of different cancers and was at the forefront of a coordinated worldwide effort towards making more molecular results from cancer analysis publicly available (Hudson et al., 2010). Other big data cancer research initiatives such as the international cancer genomics consortia, the 1000genomes[4] and the One Million Genomes projects,[5] the $10 million genome prize[6] and the remarkable drop in the cost of genome sequencing[7] will soon mean that the current paradigm in which data researchers download all the data, extract the interesting pieces and remove the rest, will no longer be feasible (Karlsson et al., 2012; Bell et al., 2009). Advances in statistical methods for analysing cancer genomics (Siegmund, 2011;

1 The Linked TCGA is available from http://aksw.org/projects/linkedtcga.
2 https://tcga-data.nci.nih.gov/tcga/
3 http://www.cancer.gov
4 http://www.1000genomes.org/
5 http://www.genomics.cn/en/navigation/show_navigation?nid=5658
6 http://in.reuters.com/article/2012/07/24/us-science-genome-prize-idINBRE86M02G20120724
7 http://www.genome.gov/sequencingcosts/

Jeong et al., 2010) further emphasizes the need to enable smooth online data collection and aggregation. As pointed out in (Chin et al., 2011) "Large-scale genome characterization efforts involve the generation and interpretation of data at an unprecedented scale, which has brought into sharp focus the need for improved information technology infrastructure and new computational tools to render the data suitable for meaningful analyses."

TCGA data has been widely used in the literature (over 350 publications[8]), but mostly in its raw form and without integration beyond a single type of molecular information (Noushmehr et al., 2010; Robinson et al., 2011; Kim et al., 2013; Hsu et al., 2012). Deus et al. (Deus et al., 2010) developed an infrastructure using Simple Sloppy Semantic Database (S3DB) management model to expose clinical, demographic and molecular data elements generated by TCGA as a SPARQL endpoint. More recently, Robbins et. al (Robbins et al., 2013) developed an engine to continuously index and annotate the TCGA files using JavaScript in conjunction with RDF, and the SPARQL query language. However, both Deus et al. (2010) and Robbins et al. (2013) provide only file level provenance annotations without providing structured access to actual contents of the files.

A scalable and robust solution is therefore a critical requirement, whereby researchers can obtain the slice of the big data they are interested in by submitting a structured query to a federated service. In addition to the very large semi-structured experimental results datasets available through TCGA and related projects, there is a significant amount of unstructured and structured biomedical data available on the web, which is critical towards annotating and integrating those experimental results. Remote query processing and virtual data integration, i.e. transparent on-the-fly-view creation for the end user, can provide a scalable solution to both challenges. Currently, due to the majority of data being available in text form, it is impossible to query the contents of a particular file or to enable virtual data integration from TCGA data sources. Indeed, the growth of TCGA initiative should also be considered for a scalable solution.[9] We addressed this problem by applying Semantic Web technologies to semi-structured level 3 TCGA data. We converted this data into resource description framework (RDF) data and linked it to the Linked Open Data Cloud so as to make it easy to query. The data can be accessed freely via SPARQL endpoints.

25.2 CONVERSION TO RDF AND LINKING

In this section, we explain the text to RDF conversion process and the linking of the resulting RDF files to the LOD Cloud.

TCGA RDFization

The TCGA text to RDF conversion process is shown in Figure 25.1. Given a TCGA text file, the Data Refiner selects the specific fields[10] necessary for traditional molecular analysis algorithms. This step is necessary to restrict the size of the resulting RDF according to what we expect will be the most useful results. Finally, the re-

8 TCGAPublications:http://cancergenome.nih.gov/researchhighlights/leadershipupdate/ZhangTCGAStats

9 http://tcga.github.io/Roadmap

10 https://code.google.com/p/topfed/wiki/SelectedFields

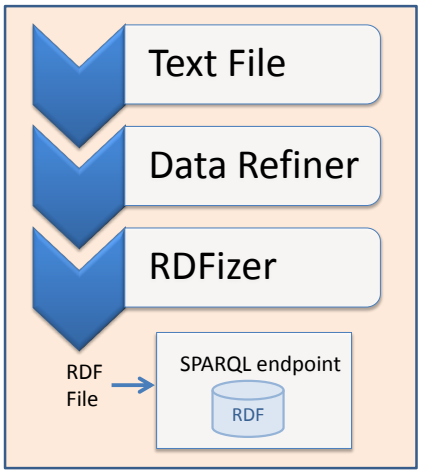


Figure 25.1: TCGA text to RDF conversion process

Tumor Type	Original Size(GB)	Refined Size (GB)	RDFized Size (GB)	Triples (Million)
Cervical (CESC)	8.75	2.44	8.86	400.19
Rectal adenocarcinoma (READ)	8.07	2.25	9.04	413.31
Papillary Kidney (KIRP)	10.40	2.90	10.4	469.65
Bladder cancer (BLCA)	12.16	3.39	12.3	556.38
Acute Myeloid Leukemia (LAML)	14.85	4.14	15.1	684.05
Lower Grade Glioma (LGG)	17.08	4.76	17.1	778.82
Prostate adenocarcinoma (PRAD)	18.05	5.03	18.1	821.01
Lung squamous carcinoma (LUSC)	20.63	5.75	20.5	927.08
Cutaneous melanoma (SKCM)	23.22	6.47	23.2	1050.94
Head and neck squamous cell(HNSC)	27.6	7.69	27.5	1245.37

Table 25.1: Overview of the 10 smallest TCGA tumors

fined text file is send to the RDFizer which generates the resulting RDF file in N3 format so that it can be loaded into any triple store, such as Virtuoso or Sesame.

As an example of the efficient space consumption feature of our RDFization, it is worth noting that original text size (20.63 GB) from the TCGA lung tumour (LUSC) is reduced to 5.75 GB after passing through the Data Refiner and the final RDF files, after passing through RDFizer, only take 20.5 GB to represent 927 million triples. After uploading these files to a virtuoso SPARQL endpoint, the total space consumption is 54 GB. The increase in size (approx. double) is caused by the different indexes created by the virtuoso server for fast data retrieval.

The statistics of the RDFization of the top 10 tumours with the smallest data files is given in Table 25.1. Given that we have produced a total of 7.34 billion triples for these tumours, we can estimate that entire TCGA level 3 data will result in over 30 billion triples. Our Linked representation of TCGA (to the best of our knowledge) thus promises to be the largest dataset available on the LOD cloud.[11]

Linking TCGA to the Linked Open Data Cloud

The fourth design principle behind Linked Data is the provision of links to other data sources. By these means, central tasks such as cross-ontology question an-

11 http://lod-cloud.net/state/

Result	Target	Class	# links
Methylation	HGNC	Chromosomes	97,530
Methylation	OMIM	Chromosomes	14,407,269
Gene expression	HGNC	Chromosomes	86,052
Gene expression	OMIM	Chromosomes	12,535,829

Table 25.2: Links for the methylation of a single patient

Source	Target	Class	# links
DNA27	HGNC	Genes	23,181
DNA27	Homologene	Genes	27,654
DNA27	OMIM	Genes	15,171
DNA450	Homologene	Genes	489,643
DNA450	OMIM	Genes	212,284
DNA27	HGNC	Chromosomes	108,662
DNA27	OMIM	Chromosomes	16,039,535

Table 25.3: Links for the lookup files of TCGA

swering, data integration and data analytics can be facilitated. Yet, the sheer size of bio-medical knowledge available on the Linked Data Cloud and of TCGA knowledge base itself makes it impossible to use manual linking to provide such cross-knowledge-base links from TCGA to other data sources. We made use of the LIMES framework[12] to compute links between TCGA and knowledge bases. LIMES (Ngonga Ngomo, 2012a) is a framework for link discovery that provides time-efficient implementations of several string and numeric similarity and distance measures. All the TCGA experimental results are reported with regards to a gene or a chromosome. Given that genes and chromosomes have dedicated IDs that are used across several knowledge bases; we used LIMES exactMatch measure for linking. As such, we focused this work on linking patient data from TCGA (and its reported genetic results) with knowledge bases which describe genes and chromosomes. In particular, we linked TCGA to HGNC,[13] OMIM[14] and Homologene.[15] Table 25.2 and Table 25.3 provides an excerpt of the links generated for the TCGA dataset while Listing 25.1 provides an excerpt of the specifications used for linking.

TCGA Data Workflow

TCGA data can be organized as a three-layer architecture in which layer 1 contains patient data, layer 2 consists of clinical information and layer 3 contains results for different samples of a patient. Each type of data was assigned to a different class in the RDFized version as depicted in the diagram in Figure 25.2. In the next section

12 http://limes.sf.net
13 http://hgnc.bio2rdf.org/sparql
14 http://omim.bio2rdf.org/sparql
15 http://homologene.bio2rdf.org/sparql

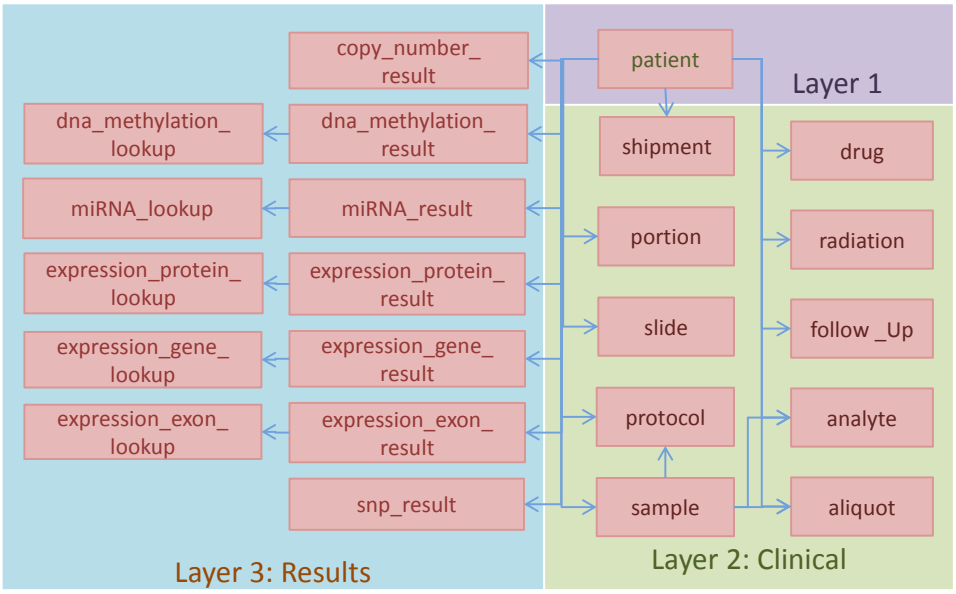


Figure 25.2: TCGA class diagram of RDFized results

we will describe some use cases where this data is applicable and illustrate the advantages of its RDFization as compared to raw experimental result text files.

Listing 25.1: Excerpt of the LIMES link specification for linking TCGA and Homologene

```
<SOURCE>
 <ID>TCGA</ID>
 <ENDPOINT>dna_methylation450_Lookup.nt</ENDPOINT>
 <VAR>?x</VAR>
 <PAGESIZE>-1</PAGESIZE>
 <RESTRICTION>?x rdf:type tcga-schema:dna_methylation450_lookup</RESTRICTION>
 <PROPERTY>tcga-schema:Gene_Symbol AS lowercase</PROPERTY>
 <TYPE>N-TRIPLE</TYPE>
</SOURCE>
<TARGET>
 <ID>homologene</ID>
 <ENDPOINT>http://homologene.bio2rdf.org/sparql</ENDPOINT>
 <VAR>?y</VAR>
 <PAGESIZE>10000</PAGESIZE>
 <RESTRICTION>?y a homologene:HomoloGene_Group</RESTRICTION>
 <PROPERTY>homologene:has_gene_symbol AS lowercase</PROPERTY>
</TARGET>
<METRIC>exactmatch(x.tcga-schema:Gene_Symbol,
   y.homologene:has_gene_symbol)</METRIC>
<ACCEPTANCE>
 <THRESHOLD>1</THRESHOLD>
 <FILE>dna_450_homologene_accepted.nt</FILE>
 <RELATION>tcga-schema:Homologene</RELATION>
</ACCEPTANCE>
```

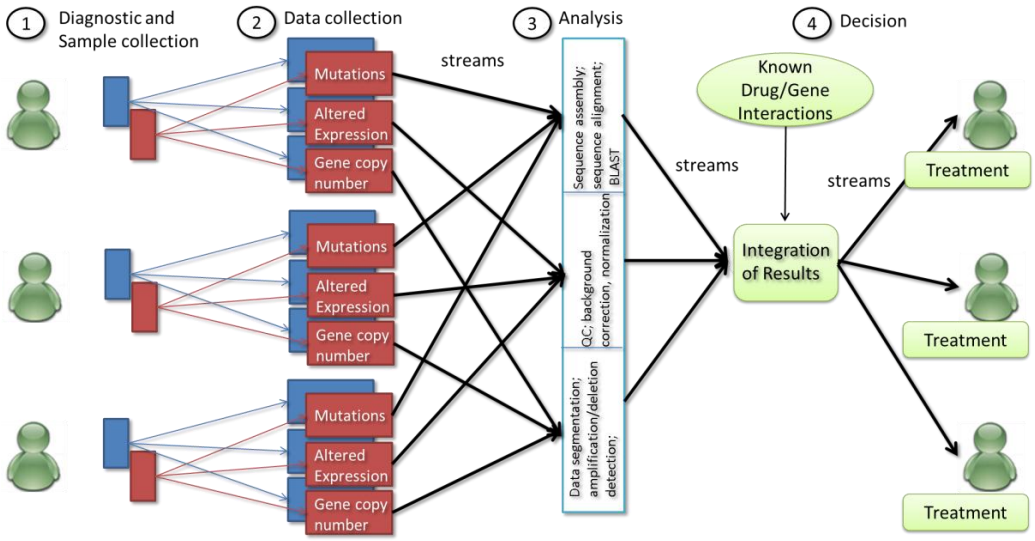


Figure 25.3: An overview of the pipeline for personalized cancer treatment

25.3 USE CASES IN CANCER TREATMENT

In Figure 25.3, we outline our final goal of linked TCGA Atlas i.e. on-the-fly data collection, analyse it and use relevant data for patient treatment. Stakeholders involved in this process include patients (the primary data providers), physicians, bioinformatics experts and statisticians. Several steps are included in this process:

1. Diagnostic and sample collection: Cancer patients around the world are asked for consent towards donating samples for the project.

2. Data collection: Each of the samples is analyzed using several molecular techniques for detecting common molecular events in cancer.

3. Analysis: Each type of data needs to be analyzed separately according to the platform used. For some of the analysis (e.g. gene expression), batches of patients must be analyzed together to enable normalization.

4. Decision: Once the results are analysed for each batch, they are integrated with other data such as known gene/drug interaction.

Currently this process is performed manually and is thus inefficient and error-prone, forcing bioinformatics experts to regularly check for new patient data or new files with data, download the data from multiple endpoints, rerun the analysis and submit the result somewhere where the physician can access and associate it with the patient diagnostic information. Due to such manual process, two critical issues occur: 1) the linking between the patient and his/her genome is lost because the clinical information cannot be made public and 2) statisticians are not encouraged to maintain provenance of models and parameters used to analyse the data, leading often to serious, and expensive mistakes. Hereafter, we describe use cases where automated pipelines will help researchers in improved and backtracked for reproducibility of results. Further, the linking of TCGA with LOD datasets will enable us to further explore the use case outcomes (e.g drugs) in the existing LOD

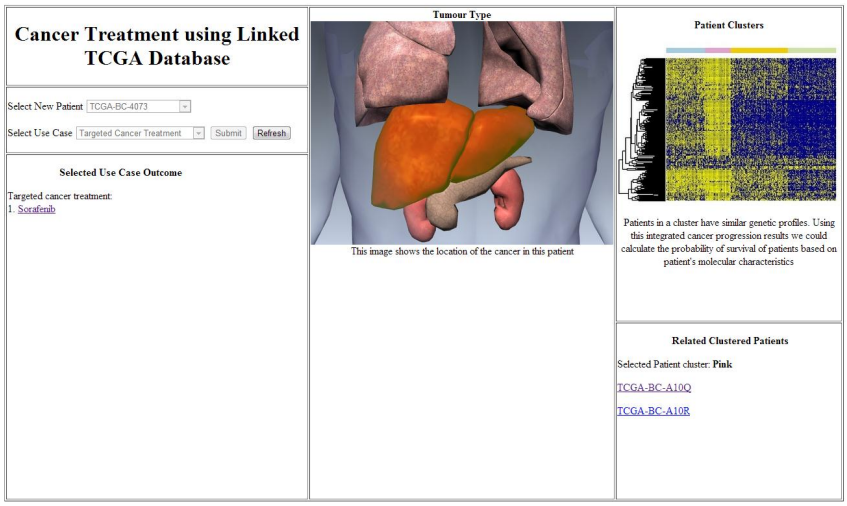


Figure 25.4: Screenshot of the Targeted Cancer Treatment

datasets such as HGNC, OMIM, Drugbank and NCBI. The demo of the use cases discussed below is available at `http://linkeddatacup-demo.deri.ie/`.

25.3.1 *Targeted Cancer Treatment*

The main question addressed in this use case is whether a specific drug can be used to treat a tumour given the genomic data of those tumour patients. An example for this use case can be seen in Breast Cancer, patients having mutations in BRCA1 and BRCA2 genes are highly susceptible to Breast Cancer and have varying treatment compared to patients without mutations in these genes. Another example in Breast Cancer is HER2-positive breast cancer, which warrants different treatment from HER2-negative breast cancers. Many such studies have been done to find clinical subtypes of different cancers for targeted treatment.

Given that these genetic mutations only occur in a handful of cases, in order for these kinds of studies to have strong statistical predictability, there is need for a very large sample size of cancer patients data - much more than what a single hospital can produce. The TCGA project aimed at assembling this very large cancer cohort but, to the best of our knowledge, the lack of a structured representation of the data proposed in the report prevented these large correlations to be derived. Given the integration of respective cancer omics data one can use different bioinformatics methods to find relevant genomic profiles in particular tumour type having specific drug effect as clinical variable of interest.

As an example, we present in Listing 25.2 a query that retrieves all patients having breast tumour, together with information about their treatment and relapse. The results of this query will be further analysed using statistical tools to find alternation in HER2 and ER genes in patients. It can be seen in Figure 25.4 that a patient is selected for targeted cancer treatment where list of drugs that are specific for such type of cancer are displayed as results. The output of this use case is based on strong correlation between the genomic data of the selected input patient and previously collected patients of same cancer. The patient clusters show the number of clinical subtypes of cancer that can be found in the collected patients genomic data. It can also be seen to which cluster (pink in this example) does the input

Listing 25.2: Use case 1,2 SPARQL query

```
SELECT ?patient ?mean
WHERE
{
        ?uri tcga:tumour_type "BRCA".
        ?uri tcga:bcr_patient_barcode ?patient.
        ?patient rdf:type  tcga:expression_gene_results.
        ?patient tcga:gene_symbol "HER2","ER".
        ?patient tcga:scaled_estimate ?mean
}
```

Listing 25.3: Querying LOD DrugBank

```
SELECT ?drugname
WHERE
{
        ?patient rdf:type tcga:expression_gene_results .
        ?patient tcga:gene_symbol ?targetname .
        ?patient tcga:scaled_estimate ?mean.
        FILTER (?mean > Threshold)
        ?drug drugbank:target ?target.
        ?drug drugbank:genericName ?drugname .
        ?target drugbank:synonym ?targetname .
        FILTER REGEX (?targetname, "HER2|estrogenreceptor|ERBB2", "i")
}
```

patient belong. Based on this information specific drug (i.e Sorafenib) is suggested for that patient's treatment. The information about the selected drug can further be explored by using the LOD datasets such as DrugBank as shown in Figure 25.4.

25.3.2 *Mechanism-based Treatment*

The main question addressed in this use case is whether a combination of drugs can be applied to treat a specific tumour effectively. Cancer can be regarded as a series of abberant genetic events leading to an uncontrollable cell growth. As mentioned in section 3.1, there are drugs specific for certain genetic events, which can be prescribed to patients differentially, making their treatment personalized. Furthermore, it is often the case that patients on one drug often relapse (because the cancer has become resistant to that drug). Thus, a combination of drugs - either prescribed together or in sequence - might be necessary for effective treatment of cancer. The patient list from statistical analysis of use case 1 results can be used to detect which patients are sensitive to the drug Transtuzumab (which targets the HER2 gene) and intercept with patients that are sensitive to the drug Tamoxifen (which targets the gene estrogen receptor) using the information from other LOD sources targeting drugs such as DrugBank as shown in Listing 25.3. The threshold in Listing 25.3 is obtained from the statistical analysis of use case 1 results. To explore such areas, integration of cancer omics data such as provided here was of great importance to produce statistically significant results.

Listing 25.4: Use case 3 SPARQL query

```
SELECT ?patient ?mean
WHERE
{
        ?uri tcga:tumour_type "BRCA".
        ?uri tcga:bcr_patient_barcode ?patient.
        ?patient rdf:type  tcga:clinical.
        ?patient tcga:tumour_stage ?tumour_stage.
        ?patient tcga:age_at_initial_patalogical_diagnosis ?age.
        ?patient tcga:relevant_biomarker "BRCA1","CDKN2A","CDH1".
        ?patient tcga:beta_value ?mean
}
```

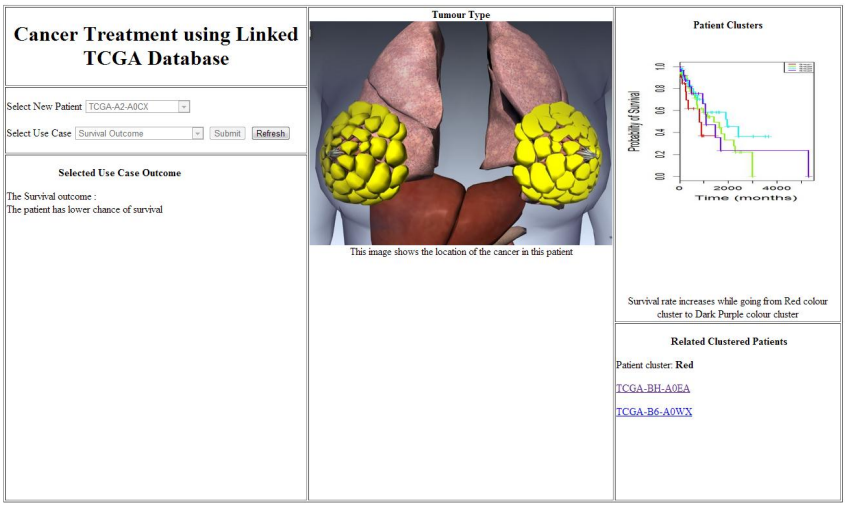


Figure 25.5: Screenshot of the Survival Outcome

25.3.3 *Survival Outcome*

The main question addressed in this use case is whether a mathematical model can be built on the patients tumour omics and clinical data in order to detect signs of tumour given the genomic profile of a future patient. It is well known that the treatment of early stage tumours has a much higher success rate than that of late-stage tumours. The classification of tumour patients based on the genetic biomarkers along with patients cancer omics and clinical data can be a powerful predictive tool with increasing tumour patients sample size. For this use case, the query given in Listing 25.4 selects patients with available tumour stage along with relevant clinical variables and selects methylation biomarkers which are correlated to the tumour stage.

The results from the Listing 25.4 are further sent for analysis tools to classify the patients in to relevant clinical clusters based on the tumour stage. The need for integrating the tumor patients' data for this use case is fulfilled by this project. It can be seen in Figure 25.5 that patients are divided in to clusters based on statistical modeling of gene expression data to patient survival time. Different clusters indicate for different survival times and patients in a certain cluster have more or less similar tumor stage. The output of this use case will be predicting the sur-

property	value
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://tcga.deri.ie/schema/patient
http://tcga.deri.ie/schema/bcr_patient_barcode	TCGA-CU-A0YN
http://tcga.deri.ie/schema/weight	67.4
http://tcga.deri.ie/schema/anatomic_organ_subdivision	Bladder, NOS
http://tcga.deri.ie/schema/diagnosis_subtype	Non-Papillary
http://tcga.deri.ie/schema/prior_diagnosis	NO
http://tcga.deri.ie/schema/days_to_death	393
http://tcga.deri.ie/schema/date_of_form_completion	2011-1-5
http://tcga.deri.ie/schema/vital_status	DECEASED
http://tcga.deri.ie/schema/age_at_initial_pathologic_diagnosis	60
http://tcga.deri.ie/schema/bcr_patient_uuid	679a6869-2ce9-4472-8db1-8869e2c1a440
http://tcga.deri.ie/schema/date_of_initial_pathologic_diagnosis	2005-00-00
http://tcga.deri.ie/schema/days_to_birth	-21927
http://tcga.deri.ie/schema/days_to_initial_pathologic_diagnosis	0
http://tcga.deri.ie/schema/ethnicity	NOT HISPANIC OR LATINO

Figure 25.6: Screenshot of the clinical TCGA related patient breast cancer data

vival rates for the given input patient based on the patient's gene expression data. The chance of survival depends upon the cluster the input patient belongs; with red cluster has the lowest and purple has the highest chance of survival. In the demo screen shot, the input patient belongs to red cluster indicating that it has lowest chance of survival. Furthermore, patients who belong to the same category of lower survival rates are shown and the clinical data of these patients can be seen in Figure 25.6 which has details of patients cancer type, drugs used, date of admission and so on.

26 LINKED STATISTICAL DATA AND METADATA EXCHANGE

PREAMBLE

In addition to being used in use cases such as benchmarking and knowledge extraction, LIMES has been used during the creation of a large number of datasets. In this and the subsequent chapters, we present three of these datasets. In this chpater, we focus on LinkedSDMX, a dataset on statistical data. The content of this chapter is taken from (Capadisli et al., 2015). The author co-supervised the work presented herein.

26.1 INTRODUCTION

While access to statistical data in the public sector has increased in recent years, a range of technical challenges makes it difficult for data consumers to tap into this data at ease. These are particularly related to the following two areas:

- Automation of data transformation of data from high profile statistical organizations.
- Minimization of third-party interpretation of the source data and metadata and lossless transformations.

Development teams often face low-level repetitive data management tasks to deal with someone else's data. Within the context of Linked Data, one aspect is to transform this raw statistical data (e.g., SDMX-ML) into an RDF representation in order to be able to start tapping into what's out there in a uniform way.

The contributions of this article are two-fold. We present an approach for transforming SDMX-ML based on XSLT 2.0 templates and showcase our implementation which transforms SDMX-ML data to RDF/XML. Following this, SDMX-ML data from Organisation for Economic Co-operation and Development (OECD)[1], Bundesamt fÃŒr Statistik (BFS, Swiss Federal Statistical Office)[2], Food and Agriculture Organization of the United Nations (FAO)[3], and European Central Bank (ECB)[4] are retrieved, transformed and published as Linked Data.

26.2 BACKGROUND

As pointed out in Statistical Linked Dataspaces (Capadisli, 2012), what linked statistics provide, and in fact enable, are queries across datasets: Given that the dimension concepts are interlinked, one can learn from a certain observation's dimension value, and enable the automation of cross-dataset queries.

Moreover, a number of approaches have been undertaken in the past to go from raw statistical data from the publisher to linked statistical data, as discussed in

1 http://www.oecd.org/
2 http://www.bfs.admin.ch/
3 http://www.fao.org/
4 http://www.ecb.int/

great detail in Official statistics and the Practice of Data Fidelity (Richard Cyganiak, 2011). These approaches go from retrieval of the data by majority; in tabular formats: Microsoft Excel or CSV, tree formats: XML with a custom schema, SDMX-ML, PC-Axis, to transformation into different RDF serialization formats (Hausenblas et al., 2012). As far as graph formats go, majority of datasets in those formats are not published by the owners. However, there are number of statistical linked dataspaces in the LOD Cloud already[5].

A number of transformation efforts are performed by the Linked Data community based on various formats. For example, the World Bank Linked Dataspace[6] is based on custom XML that the World Bank[7] provides through their APIs with the application of XSL Templates. The Transparency International Linked Dataspace's data is based on CSV files with the transformation step through Google Refine[8] and the RDF Extension[9]. That is, data sources provide different data formats for the public, with or without accompanying metadata e.g., vocabularies, provenance. Hence, this repetitive work is no exception to Linked Data teams as they have to constantly be involved either by way of hand-held transformation efforts, or in best-case scenarios, it is done semi-automatically. Currently, there is no automation of the transformation step to the best of our knowledge. This is generally due to the difficulty of the task when dealing with the quality and consistency of the statistical data that is published on the Web, as well as the data formats that are typically focused on consumption. Although SDMX-ML is the primary format of the high profile statistical data organizations, it is yet to be taken advantage of.

26.3 SDMX-ML TO LINKED DATA

Recently, SDMX was approved by ISO as an international standard: ISO 17369:2013[10]. It is a standard which provides the possibility to consistently carry out data flows between publishers and consumers. SDMX-ML (using XML syntax) is considered to be the industry standard for expressing statistical data. It has a highly structured mechanism to represent statistical observations, classifications, and data structures. Organizations supporting SDMX are Bank for International Settlements (BIS)[11], Organisation for Economic Co-operation and Development (OECD), United Nations (UN)[12], European Central Bank (ECB), World Bank, International Monetary Fund (IMF)[13], Food and Agriculture Organization of the United Nations (FAO) and Eurostat[14].

We argue that high-fidelity statistical data representation in Linked Data should take advantage of SDMX-ML as it is widely adopted by data producers with rich data about our societies, making the need for transforming SDMX-ML to RDF and publishing accompanying Linked Dataspaces of paramount importance.

5 http://lod-cloud.net/
6 http://worldbank.270a.info/
7 http://worldbank.org/
8 http://code.google.com/p/google-refine/
9 http://refine.deri.ie/
10 http://www.iso.org/iso/catalogue_detail.htm?csnumber=52500
11 http://www.bis.org/
12 http://www.un.org/
13 http://imf.org/
14 http://epp.eurostat.ec.europa.eu/

26.3.1 *Data Sources*

As a demonstration of the SDMX-ML to RDF transformations, we selected datasets from the following organizations:

- OECD, whose mission is to promote policies that will improve the economic and social well-being of people around the world.
- BFS Swiss Statistics, due to the Federal Statistical Office's web portal offering a wide range of statistical information including population, health, economy, employment and education.
- FAO, which works on achieving food security for all to make sure people have regular access to enough high-quality food.
- ECB, whose main task is to maintain the euro's purchasing power and thus price stability in the euro area.

The OECD, FAO, and ECB datasets consisted of observational and structural data. The OECD and ECB data provided complete coverage (to the best of our knowledge), whereas FAO had partial fishery related data. BFS had all of their classifications available, with no observational data in SDMX-ML.

The architectural workflow of the dataspaces consists of data retrieval, transformations, enrichment, storage and publication. Along the way, information about provenance is incorporated in some of the phases.

26.3.2 *Data Retrieval*

As SDMX-ML publishers have their own publishing processes, availability and accessibility of the data varied. We performed a combination of HTML scraping, site search for SDMX files and data catalog retrieval to obtain the dataset codes, names, and URLs, which we then fed into a Bash script to retrieve the actual data. Details can be found in [15].

By in large, there was no need to pre-process the data as the transformation dealt with the data as it was. However, some non-vital SDMX components were omitted from the output. For instance, one type of attribute in OECD and ECB observations contained free-text as opposed to its corresponding code from a codelist. Since the RDF Data Cube required codes as opposed to free-text for dimension values, some attributes were excluded. The decision here was to trade-off some precision in favour of retaining the dataset.

26.4 PROVENANCE

26.4.1 *Provenance at Retrieval*

At the time of data retrieval, information pertaining to provenance was captured using the *PROV Ontology*[16] in order to further enrich the data. This RDF/XML document contains `prov:Activity` information which indicates the location of the

15 http://csarven.ca/linked-sdmx-data
16 http://www.w3.org/TR/prov-o/

XML document on the local filesystem. It contains other provenance data like when it was retrieved such as the tools that were used to process the data. This provenance data from retrieval may be provided to the XSL Transformer during the transformation phase and VoID enrichment.

26.4.2 *Provenance at Transformation*

Resources of type `qb:DataStructureDefinition`, `qb:DataSet`, `skos:ConceptScheme` are also typed with the `prov:Entity` class. Also properties `prov:wasAttributedTo` were added to these resources with the `creator` value which is of type `prov:Agent` obtained from XSLT configuration. There is a unique `prov:Activity` for each transformation, and it has a `dcterms:title`, and contains values for `prov:startedAtTime`, `prov:wasAssociatedWith` (the creator), `prov:used` (i.e., source XML, XSL to transform) to what was `prov:generated` (and source data URI that it was derived from). It also declares `dcterms:license` where value taken from XSLT configuration. The provenance document from the retrieval phase may be provided to the transformer. In this case, it establishes a link between the current provenance activity (i.e., the transformation), with the earlier provenance activity (i.e., the retrieval) using the `prov:wasInformedBy` property.

26.4.3 *Provenance at Post-processing*

The post-processing step for provenance is intended to retain provenance data for future use. As datasets get updated, it is important to preserve information about past activities by way of exporting all instances of the `prov:Activity` class from the RDF store. Activities are unique artifacts, on a conceptual level as well as with regard to referencing them. Since one of the main concerns of provenance is to keep track of activities, this post-processing step also allows us to retain a historical account of all activities during the data lifecycle, and to preserve all previously published URIs (cf. Cool URIs don't change[17]).

26.5 DATA MODELING

In this section we go over several areas which are at the heart of representing SDMX-ML data as Linked Data. The approach taken was to provide a level of consistency for data consumers and developers.

26.5.1 *Vocabularies*

In addition to RDF, RDFS, XSD, OWL, the RDF Data Cube vocabulary (Tennison et al., 2012) is used to describe multi-dimensional statistical data, SDMX-RDF is used for the statistical information model. PROV-O is used for capturing provenance data. SKOS and XKOS to cover concepts, concept schemes and their relationships.

17 http://www.w3.org/Provider/Style/URI.html

26.5.2 *Versioning*

SDMX data publishers version their classifications and the generated cubes refer to particular versions of those classifications. Consequently, versions need to be explicitly part of classification URIs in order to uniquely identify them. Although including version information in the URI is disputed by some authors, we deem it is as a good practice for identifying different concepts and data structures. Jeni Tennison et al discussed Versioning URIs[18], and concluded that there was no one-size-fits all solution. An alternative approach using named graphs for a series of changes was proposed in Linking UK Government Data (Sheridan and Tennison, 2010).

26.5.3 *URI Patterns*

An outline for the URI patterns is given in Table 26.1. `authority` is replaced with the domain (see also: Agency identifiers and URIs) followed by `class`, `code`, `concept`, `dataset`, `property`, `provenance`, or `slice` for each prominent area in SDMX structures. These tokens as well as `/` which is used to separate the dimension concepts in URIs can be configured in our toolkit. In order to construct the URIs for the above patterns, some of the data values are normalized to make them URI safe but not altered in other ways (e.g., lower-casing). The rationale for this was to keep the consistency of terms in SDMX and RDF.

Table 26.1: URI patterns

Entity type	URI Pattern
qb:DataStructureDefinition	http://{authority}/structure/{KeyFamilyID}
qb:DataSet	http://{authority}/dataset/{datasetID}
qb:Observation	http://{authority}/dataset/{datasetID}/{dimension-1}/../{dimension-n}
qb:Slice	http://{authority}/slice/{KeyFamilyID}/{dimension-1}/../{dimension-n-no-FREQ}
skos:Collection	http://{authority}/code/{version}/{hierarchicalCodeListID}
	http://{authority}/code/{version}/{hierarchyID}
sdmx:CodeList	http://{authority}/code/{version}/{codeListID}
skos:ConceptScheme	http://{authority}/concept/{version}/{conceptSchemeID}
skos:Concept, sdmx:Concept	http://{authority}/code/{version}/{codeListID}/{codeID}
	http://{authority}/concept/{version}/{conceptSchemeID}/{conceptID}
owl:Class, rdfs:Class	http://{authority}/class/{version}/{codeListID}
rdf:Property	http://{authority}/property/{conceptID}
qb:DimensionProperty	http://{authority}/property/{conceptID}
qb:MeasureProperty	http://{authority}/property/{conceptID}
qb:AttributeProperty	http://{authority}/property/{conceptID}

26.5.4 *Datatypes*

XSD datatypes are assigned to literals are based on the value of the measure component (e.g., decimal, year). In the absence of this datatype, observation values are checked whether they can be casted to `xsd:decimal`. Otherwise, they are left as plain literals.

18 http://www.jenitennison.com/blog/node/112

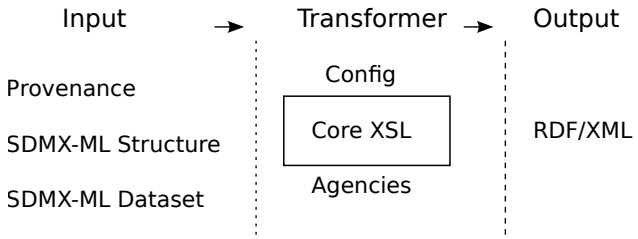


Figure 26.1: Transformation process

26.6 LINKED SDMX DATA TRANSFORMATION

The Linked SDMX XSLT 2.0 templates and scripts[19] are developed to transform SDMX-ML data and metadata to RDF/XML. Its goals are:

- To improve access and discovery of cross-domain statistical data.
- To perform the transformation in a lossless and semantics preserving way.
- To support and encourage statistical agencies to publish their data using RDF and integrating the transformation into their workflow.

The key advantage of this transformation approach is that additional interpretations are not required by the data modeler especially in comparison to alternative transformation (e.g., CSV or XML to RDF serialization). Since the SDMX-RDF vocabulary is based on SDMX-ML standard, and the RDF Data Cube vocabulary is closely aligned with the SDMX information model, the transformation is to a large extent a matter of mapping the source SDMX-ML data to its counter parts in RDF.

26.6.1 *Features of the transformation*

- Transformation of SDMX KeyFamilies, ConceptSchemes and Concepts, CodeLists and Codes, Hierarchical CodeLists, and DataSets.
- Configurability for SDMX publisher's needs.
- Detection and referencing CodeLists and Codes of external agencies.
- Support of interlinking publisher-specific annotation types.
- Support for omission of components.
- Inclusion of provenance data.

26.6.2 *Configuration*

The requirements for the Linked SDMX toolkit are an XSLT 2.0 processor to transform, and optionally to configure some of the settings in the transformation. In sequel some of they key features are described in more detail.

19 https://github.com/csarven/linked-sdmx

AGENCY IDENTIFIERS AND URIS An RDF file is used to lookup information on maintenance agencies (i.e., the data owner and publisher). It includes maintenance agencies' identifiers in the SDMX Registry, as well as their base URI. It allows to look up base URIs using the agency identifier. For example, Listing 26.1 shows a coded property that is used by European Central Bank to associate a code list defined by Eurostat as an external agency:

Listing 26.1: Referencing external agencies.

```
<http://ecb.270a.info/property/OBS_STATUS>
 <http://purl.org/linked-data/cube#codeList>
 <http://eurostat.270a.info/code/1.0/CL_OBS_STATUS>
```

We decided to avoid re-defining metadata from external agencies, since the owners of the data would define them under their authority. If the agency identifier is `SDMX` the corresponding URIs from the SDMX-RDF vocabulary are used instead.

URI CONFIGURATIONS Separate base URIs can be set for classes, codelists, concept schemes, datasets, slices, properties, provenance, as well as for the location of the source data for transformations. The value for `uriThingSeparator` (e.g., `/`), sets the delimiter for separating the "thing" from the rest of the URI. This is typically either a `/` or `#`. Similarly, `uriDimensionSeparator` can be set to separate dimension values used in RDF Data Cube observation URIs. Each observation requires a unique URI construction. One simple and user-friendly approach is to construct URIs by using URI-safe dimension values as tokens separated by the `uriDimensionSeparator`. Listing 26.2 shows an example observation URI with `/` as `uriDimensionSeparator`.

Listing 26.2: Example observation URI

```
http://{authority}/dataset/HEALTH_STAT/EVIEFE00/EVIDUREV/AUS/1960
```

DEFAULT LANGUAGE From the configuration, it is possible to assign a default language on `skos:prefLabel` and `skos:definition` property values, when language is not originally set for a data item. Default language may also be applied in the case of SDMX Annotations.

INTERLINKING SDMX ANNOTATIONS The conventions in annotations typically differ from one SDMX publisher to another as there is no standardization. In order to retain this valuable information, the configuration file allows publishers to define the way annotations should be transformed. This is accomplished by defining the annotation types that should be interlinked or described with, by providing the range i.e., either an URL or a literal. The predicate to connect both resources are also defined here.

OMITTING COMPONENTS There are cases in which certain data parts contain errors. The configuration option `omitComponents` allows to omit this erroneous data without effecting other parts, as well as to abstain from making any significant assumptions or changes to the data.

Dataset	Input size	Output size	Ratio	Time
OECD	3,430 MB	23,000 MB	1:6.7	7885s
BFS	87 MB	139 MB	1:1.6	158s
FAO	902 MB	5,000 MB	1:5.5	1908s
ECB	5,670 MB	24,000 MB	1:4.2	10863s

Table 26.2: Transformation time

Dataset	# triple	# `qb:Observation`	Ratio
OECD Dataset	225M	24M	9.4:1
OECD Metadata	0.77M	N/A	N/A
BFS Metadata	1M	N/A	N/A
FAO Dataset	53M	7.2M	7.4:1
FAO Metadata	0.36M	N/A	N/A
ECB Dataset	241M	12.5M	19.3:1
ECB Metadata	0.45M	N/A	N/A

Table 26.3: Transformed data

26.7 LINKED DATASETS

This section describes the transformation result and the publication of the OECD, BFS, FAO, and ECB datasets.

26.7.1 *RDF Datasets*

The original SDMX-ML files were transformed to RDF/XML using XSLT 2.0. Saxon's command-line XSLT tool `saxonb-xslt` was used and employed as part of shell scripts to iterate through all the files in the datasets. 12 GB of memory were allocated on a machine with Linux kernel 3.2.0-33-generic running on an Intel(R) Xeon(R) CPU E5620 @ 2.40GHz. Table 26.2 provides information on datasets; input SDMX-ML size, output RDF/XML size, their size difference in ratio, and the total amount transformation time. Table 26.3 summarizes the transformed data; number of triples it contains, as well as the number of `qb:Observation`, and the ratio. Table 26.4 provides further statistics on prominent resources. It gives a contrast between the classifications and the dataset.

26.7.2 *Interlinking*

SDMX concept schemes and code lists are two valuable artifacts that are used by data owners to precisely denote the meaning of observational data. The concepts and codes within are also reused by external agencies by way of referring to their unique identifiers. Thus, the interlinking phase that was undertaken for the datasets is complimentary to referencing external code lists as discussed in Agency

Resource	OECD	BFS	FAO	ECB
skos:ConceptScheme	1,212	185	32	149
skos:Concept	43,368	106,233	28,115	54,389
rdf:Property	126	0	12	209
qb:Observation	24,381,106	0	7,186,764	12,513,494

Table 26.4: Resource counts

identifiers and URIs. Initial interlinking is done among the classifications themselves in the datasets. The OECD classifications in particular contained highly similar codes (in some cases the same) throughout its code lists. Hence, the majority of the codes were interlinked with one another using the property `skos:exactMatch`. Further interlinking was performed among the datasets themselves as well as with other datasets using the LIMES link discovery framework (Ngonga Ngomo, 2012b), including: DBpedia[20], World Bank[21], Transparency International[22], and EUNIS[23]. Table 26.5 describes the interlinking between the datasets. Figure 26.2 provides an overview on the complete connectivity of a concept. This comprises linking internally, externally, and with sdmx-codes where applicable, as well as the interlinking with external concepts.

Source	Target	Entity type	Link relation	Count
OECD	World Bank	skos:Concept, dbo:Country	skos:exactMatch	3,487
OECD	Transparency International	skos:Concept, dbo:Country	skos:exactMatch	3,335
OECD	DBpedia	skos:Concept, dbo:Country	skos:exactMatch	3,391
OECD	BFS	skos:Concept, code:CL_STATES_AND_TERRITORIES	skos:exactMatch	3,383
BFS	World Bank	code:CL_STATES_AND_TERRITORIES, dbo:Country	skos:exactMatch	185
BFS	DBpedia	skos:Concept, dbo:Country	skos:exactMatch	261
FAO	DBpedia	skos:Concept, dbo:Species	skos:exactMatch	673
FAO	EUNIS	skos:Concept, e:sameSynonymFIFAO	skos:exactMatch	359
ECB	World Bank	skos:Concept, dbo:Country	skos:exactMatch	188
ECB	Transparency International	skos:Concept, dbo:Country	skos:exactMatch	167
ECB	DBpedia	skos:Concept, dbo:Country	skos:exactMatch	239
ECB	BFS	skos:Concept, code:CL_STATES_AND_TERRITORIES	skos:exactMatch	221

Table 26.5: Links between datasets

26.7.3 *RDF Data Storage*

Apache Jena's TDB storage system[24] is used to load the RDF data using the TDB incremental `tdbloader` utility. `tdbstats`, the tool for TDB Optimizer is executed after a complete load to internally update the resource counts for query optimization.

20 http://dbpedia.org/
21 http://worldbank.270a.info/
22 http://transparency.270a.info/
23 http://eunis.eea.europa.eu/
24 http://incubator.apache.org/jena/documentation/tdb/

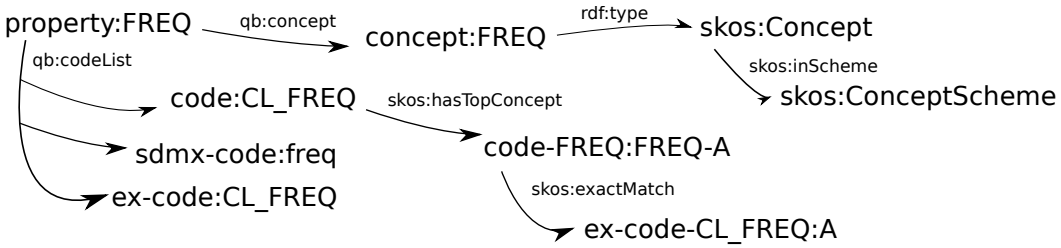


Figure 26.2: SDMX Concept links

Each dataset was imported into its own `NAMED GRAPH` in the store. Given the significant load speed on an empty database, N-Triple files were ordered from largest to smallest, and then loaded.

26.8 PUBLICATION

26.8.1 *Dataset Discovery and Statistics*

The Vocabulary of Interlinked Datasets (VoID)[25] file gives an overview of the dataset, for example, what it contains, ways to access or query the dataset. Each dataspace contains files accessible through their `.well-known/void` locations. Each OECD, BFS, FAO, and ECB VoID[26] contains locations to RDF datadumps, named graphs that are used in the SPARQL endpoint, used vocabularies, size of the datasets, interlinks to external datasets, as well as the provenance data which was gathered through the retrieval and transformation process. The VoID files were generated automatically by first importing the LODStats (Auer et al., 2012) information into a `graph/void` named graph, and then executing a SPARQL `CONSTRUCT` query to include all triples as well as relevant additional information from other graphs.

26.8.2 *User Interface*

The HTML pages are generated by the Linked Data Pages[27] framework, which employs Moriarty[28], Paget[29], and ARC2.[30]

26.8.3 *SPARQL Endpoint*

Apache Jena Fuseki[31] is used to run the SPARQL server for the datasets. SPARQL endpoints are publicly accessible and read only at their respective `/sparql` and `/query` locations for OECD, BFS, FAO, and ECB.[32] Currently, 12 GB of memory is allocated for the single Fuseki instance serving all datasets.

25 http://www.w3.org/TR/void/
26 http://{ oecd | bfs | fao | ecb }.270a.info/.well-known/void
27 https://github.com/csarven/linked-data-pages
28 http://code.google.com/p/moriarty/
29 http://code.google.com/p/paget/
30 https://github.com/semsol/arc2
31 http://incubator.apache.org/jena/documentation/serving_data/
32 http://{oecd | bfs | fao | ecb }.270a.info/sparql

26.8.4 *Data Dumps*

The data dumps for the datasets are available from their respective `/data/` directories: OECD, BFS, FAO, and ECB.[33] Additionally, they are referenced in the VoID files and from the Data Hub[34] entries.

26.8.5 *Source Code*

The Linked SDMX toolkit and for retrieval and data loading to the RDF store for OECD, BFS, FAO, and for ECB[35] is available at GitHub[36] using the Apache License 2.0.[37]

26.8.6 *Data License*

All published Linked Data adheres to original data publisher's data license and terms of use. Additionally attributions are given on the websites. The *Linked Data* version of the data is licensed under CC0 1.0 Universal (CC0 1.0) Public Domain Dedication.[38]

26.9 CONCLUSIONS

With this work we provided an automated approach for transforming statistical SDMX-ML data to Linked Data in a single step. As a result, this effort helps to publish and consume large amounts of quality statistical Linked Data. Its goal is to shift focus from mundane development efforts to automating the generation of quality statistical data. Moreover, it facilitates to provide RDF serializations alongside the existing formats used by high profile statistical data owners. Our approach to employ XSLT transformations does not require changes to well established workflows at the statistical agencies.

One aspect of future work is to improve the SDMX-ML to RDF transformation quality and quantity. Regarding quality, we aim to test our transformation with further datasets to identify shortcomings and special cases being currently not yet covered by the implementation. Also, we plan the development of a coherent approach for (semi-)automatically interlinking different statistical dataspaces, which establishes links on all possible levels (e.g. classifications, observations). With regard to quantity, we plan to publish statistical dataspaces for Bank for International Settlements (BIS), International Monetary Fund (IMF), World Bank and Eurostat based on SDMX-ML data.

The current transformation is mostly based on the generic SDMX format. Since some of the publishers make their data available in compact SDMX format, the transformation toolkit has to be extended. Alternatively, the compact format can be transformed to the generic format first (for which tools exist) and then Linked SDMX transformations can be applied. Ultimately, we hope that Linked Data pub-

33 http://{ oecd | bfs | fao | ecb }.270a.info/data/
34 http://datahub.io/
35 https://github.com/csarven/{ oecd | bfs | fao | ecb }-linked-data
36 https://github.com/csarven/linked-sdmx
37 http://www.apache.org/licenses/LICENSE-2.0.html
38 http://creativecommons.org/publicdomain/zero/1.0/

lishing will become a direct part of the original data owners workflows and data publishing efforts. Therefore, further collaboration on this will expedite the provision of uniform access to statistical Linked Data.

27

MULTILINGUAL RESOURCE FOR NATURAL-LANGUAGE PROCESSING

PREAMBLE

This chapter continues to demonstrate the versatility of the LIMES framework by presenting a linguistic resource that was built using LIMES. In particular, we describe the Semantic Quran dataset, a multilingual RDF representation of translations of the Quran. The dataset was created by integrating data from two different semi-structured sources and aligned to an ontology designed to represent multilingual data from sources with a hierarchical structure. The resulting RDF data encompasses 43 different languages which belong to the most under-represented languages in the Linked Data Cloud, including Arabic, Amharic and Amazigh. The content of the chapter is taken from (Sherif and Ngonga Ngomo, 2015). The author created some of the LIMES specifications, co-wrote the paper and supervised the work.

27.1 INTRODUCTION

Over the last years, the Linked Open Data (LOD) movement has gained significant momentum (Auer et al., 2013a). A large number of datasets was extracted from sources as different as Wikipedia infoboxes and curated bio-medical databases. Still, most of the datasets in the Linked Data Cloud contain only English labels and fail to represent the diversity of languages used across the Web.[1] Yet, a more multilingual Linked Data Cloud would represent a tremendous resource that can be used for novel knowledge extraction techniques and more broadly for novel natural-language processing (NLP) approaches. For example, novel NLP approaches for minority languages could be developed by reusing information available across the different languages (Somers, 1997). Moreover, a structured representation of corpora would improve their use in applications such as the specification of templates for question answering (Unger et al., 2012) or the efficient merging with other resources (Hellmann, 2010).

In this paper, we present the *Semantic Quran dataset*. The Semantic Quran dataset consists of all chapters of the Quran in 43 different languages including rare languages such as Divehi, Amazigh and Amharic. The data included in our dataset was extracted from two semi-structured sources: the Tanzil project and the Quranic Arabic Corpus (cf. Section 27.4). We designed an ontology for representing this multilingual data and their position in the Quran (i.e., numbered chapters and verses). In addition to providing aligned translations for each verse, we provide morpho-syntactic information on each of the original Arabic terms utilized across the dataset. Moreover, we linked the dataset to three versions of *Wiktionary* as well

1 From the 315 datasets analyzed by the LodStats framework (http://stats.lod2.eu), 128 datasets provide English labels. French, the second most popular language in the LOD Cloud, is used in only 15 (approximately 4.8%) of the datasets. Most other languages occur in at most one dataset.

as *DBpedia* and ensured therewith that our dataset abides by all Linked Data principles.[2]

In the following, we present the data sources that we used for the extraction (Section 27.2). Thereafter, we give an overview of the ontology that underlies our dataset (Section 27.3). Section 27.4 depicts the extraction process that led to the population of our ontology. We present our approach to interlinking the Semantic Quran and Wiktionary in Section 27.5. Finally, we present several usage scenarios for the dataset at hand (Section 27.6).

27.2 DATA SOURCES

Two web resources were used as raw data sources for our dataset. The first web resource is the data generated by the *Tanzil Project*,[3] which consists of the original verses in Arabic as well as 42 manual translations of the entire book. Our second web resource, *the Quranic Arabic Corpus*,[4] was used to obtain morpho-syntactic information on each of the words contained in the Arabic version of the Quran.

27.2.1 *Tanzil Project*

The Tanzil Project[5] was motivated by inconsistencies across the different digital versions of the Quran. These were mainly due to missing/incorrect diacritics, Arabic text conversion problems, and missing encoding for some Arabic characters.

Tanzil was launched in early 2007 with the aim of producing a curated unicode version of the Arabic Quran text that can serve as a reliable standard text source on the web. To achieve this goal, then Tanzil team developed a three-step data quality assurance pipeline which consists of (1) an automatic text extraction of the Arabic text, (2) a rule-based verification of the extraction results and (3) a final manual verification by a group of experts.

The result of this process was a set of datasets that were made available in several versions and formats.[6] In addition to the original Arabic sources, Tanzil provides sentence-parallel translations of the Quran in 42 different languages by different translators.[7] We manually selected one translation per language for the extraction process.[8] Note that all Tanzil datasets are distributed under the terms of Creative Commons Attribution 3.0 License.[9]

27.2.2 *The Quranic Arabic Corpus Project*

The Quranic Arabic Corpus is an open-source project, which provides Arabic annotated linguistic resources which shows the Arabic grammar, syntax and morphology for each word in the Quran. This is a valuable resources for the development of NLP tools for the Arabic language, in which a single word can encompass the semantics of entire English sentences. For instance the Arabic word *"faja'alnāhum"*

2 http://www.w3.org/DesignIssues/LinkedData.html
3 http://tanzil.net/
4 http://corpus.quran.com
5 http://tanzil.net/wiki/Tanzil_Project
6 For more details on available formats and datasets, please see http://tanzil.net/download/.
7 http://tanzil.net/trans/.
8 The list of translations used can be found at http://goo.gl/s5RuI
9 http://creativecommons.org/licenses/by/3.0/

can be translated into the entire English sentence "and we made them". The compact syntax of Arabic leads to that a single word being separable into distinct morphological segments. For example, *"faja'alnāhum"* can be subdivided into:

- *fa* – a prefixed conjunction (engl. "and"),
- *ja'al* – the stem, a perfect past tense verb (engl. "made") inflected as first person masculine plural,
- *nā* – a suffixed subject pronoun (engl. "we") and
- *hum* – a suffixed object pronoun (engl. "them").

A Resource Description Framework (RDF) and Natural Language Processing Interchange Format (NIF)(Hellmann et al., 2012) representation of this rich morphology promises to further the development of integrated NLP pipelines for processing Arabic. In addition, given that this corpus was curated manually by experts, it promises to improve the evaluation of integrated NLP frameworks. We thus decided to integrate this data with the translation data available in the Tanzil datasets. Here, we used the Quranic Arabic Corpus Version 0.4[10] in its delimited text file version under the *"GNU General Public License"*.[11]

27.3 ONTOLOGY

To represent the data as RDF, we developed a general-purpose linguistic vocabulary. The vocabulary[12] was specified with the aim of supporting datasets which display a hierarchical structure. It includes four basic classes: Chapter, Verse, Word and LexicalItem.

The *Chapter class* provides the name of chapters in different languages and localization data such as chapter index and order. Additionally, the chapter class provides metadata such as the number of verses in a chapter and provenance information. Finally, the chapter class provides properties that allow referencing the verses it contains. For example each chapter provides a `dcterms:tableOfContents` for each of its verses in the form `qrn:quran<chapter>- <verse>`.

The *Verse class* contains the verse text in different languages as well as numerous localization data such as verse index and related chapter index. Additionally, this class provides related verse data such as different verse descriptions and provenance information. Finally, it contains referencing properties similar to those of chapters.

The *Word class* encompasses the next level of granularity and contains the words in the verse text in different languages as well as numerous localization data such as related verse and chapter verse indexes. Additionally, the word class provides word provenance information and some referencing properties.

Currently, the *LexicalItem class* provides morphological data on the Arabic words only. Several ontologies can be used to represent such information. In our dataset, we relied on the RDF representation of the *GOLD linguistic ontology*[13] (Farrar and Langendoen, 2003) to provide linguistic properties of lexical items such as acoustic,

10 http://corpus.quran.com/download/
11 http://www.gnu.org/licenses/gpl.html
12 http://mlode.nlp2rdf.org/datasets/qvoc.owl.ttl
13 http://linguistics-ontology.org/

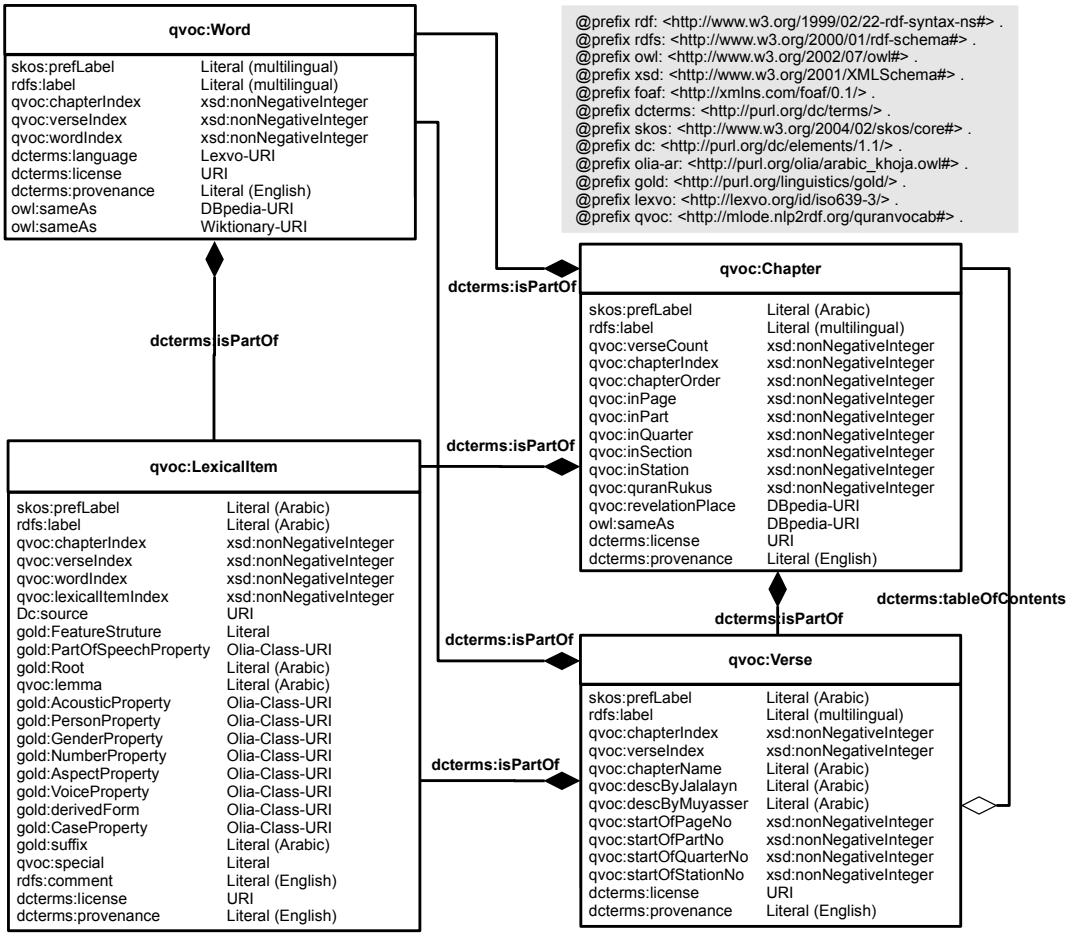


Figure 27.1: UML class diagram of the Semantic Quran Ontology

root, part of speech, gender, number, and person. We chose to use GOLD in contrast to other ontologies because it belongs to the most exhaustive ontologies for modeling linguistic properties. Thus, it will allow us to easily extend this dataset in future work. All the objects of the previously mentioned properties are URIs from the *OLIA Arabic Linguistic ontology*.[14] Analogously to the other classes, LexicalItem provides provenance information and referencing properties. A UML class diagram of the four basic ontology classes of the Semantic Quran Dataset with inter-class internal relations is shown in Figure 27.1.

27.4 EXTRACTION PROCESS

The original Tanzil Arabic Quran data and translations are published in various formats. For the sake of effectiveness, delimited text files were selected as the basis for the RDF extraction. The format of the delimited file is `chapterIndex|verse|verseText`. For example, the first verse of the first chapter of the English translation of the Quran is `1|1|In the Name of Allah, the Most Beneficent, the Most Merciful.` On the other hand, the Quranic Arabic corpus is available as tab-separated text file of the form "`LOCATION FORM TAG FEATURES`":

14 http://nachhalt.sfb632.uni-potsdam.de/owl/

Name	SemanticQuran
Example Resource	http://mlode.nlp2rdf.org/resource/semanticquran/quran1-1
Dataset dump	http://mlode.nlp2rdf.org/datasets/semanticquran.nt.gz
Sparql Endpoint	http://mlode.nlp2rdf.org/sparql
Dataset graph	http://thedatahub.org/dataset/semanticquran
Ontology	http://mlode.nlp2rdf.org/datasets/qvoc.owl.ttl
Ver. Date	29.11.2012
Ver. No	1.0
Licence	Attribution-NonCommercial-ShareAlike 3.0 Unported (CC BY-NC-SA 3.0)
CKAN	SemanticQuran

Table 27.1: Technical details of the Quran RDF dataset

- The `LOCATION` field consists of 4-part numbering scheme of the form (Chapter : Verse : Word : Segment). For example, the first segment of the first word of the first verse of the first chapter has the form `(1:1:1:1)`.
- The `FORM` field contains the text of the current segment in the Extended *Buckwalter transliteration*.[15] For example the corresponding `FORM` to `(1:1:1:1)` is `bi` (engl. "In").
- The `TAG` field contains the part-of-speech tag for the current segment. For example the corresponding `TAG` to `(1:1:1:1)` is `p` which stands for preposition.
- The `FEATURES` field contains a complete morphological analysis of the current segment such as root, case and person-number-gender properties. For example the corresponding `FEATURES` to `(1:1:1:1)` is `PREFIX|bi+` which stands for preposition prefix ("by", "with", "in") with acoustic property ”bi“.

Given the regular syntax used in the text file corpus at hand, we were able to carry out a one-to-one mapping of each fragment of the input text file to resources, properties or data types as explicated in the ontology shown in Figure 27.1. We relied on the *Apache Jena Framework*[16] for the conversion. The part-of-speech information and morphological characteristics of each segment of the Arabic Quranic Corpus were extracted and integrated with the words found in the Tanzil dataset. The merged data is now available in the RDF format. In order to simplify the interoperability of the generated dataset, we followed the specifications of the NIF. Currently, the original Arabic and four different translations of the Quran (Arabic, English, German, French and Russian) abide by the NIF formalization. Details of the Semantic Quran dataset CKAN entry, its SPARQL endpoint, version and license are listed in Table 27.1.

27.5 LINKING

We aimed to link our dataset with as many data sources as possible to ensure maximal reusability and integrability in existing platforms. We have generated links to 3 versions of the RDF representation of Wiktionary as well as to DBpedia. All links were generated by using the LIMES framework (Ngonga Ngomo, 2012b). The link

15 The Buckwalter transliteration uses ASCII characters to represent the orthography of the Arabic language. For the conversion table, see http://www.qamus.org/transliteration.htm

16 http://jena.apache.org/

specification used was essentially governed by fragments similar to that shown in Listing 27.1. The basic intuition behind this specification is to link words that are in a given language in our dataset to words in the same language with exactly the same label. We provide 7617 links to the English version of DBpedia, which in turn is linked to non-English versions of DBpedia. In addition, we generated 7809 links to the English, 9856 to the French and 1453 to the German Wiktionary. Links to further versions of DBpedia and Wiktionary will be added in the future.

Listing 27.1: Fragment of the link specification to the English Wiktionary

```
<SOURCE>
<ID>quran</ID>
<ENDPOINT>http://mlode.nlp2rdf.org/sparql</ENDPOINT>
<VAR>?x</VAR>
<PAGESIZE>-1</PAGESIZE>
<RESTRICTION>?x a qvoc:Word</RESTRICTION>
<PROPERTY>rdfs:label AS lowercase->nolang
RENAME label </PROPERTY>
</SOURCE>
<TARGET>
<ID>wiktionary</ID>
<ENDPOINT>http://wiktionary.dbpedia.org/sparql
 </ENDPOINT>
<VAR>?y</VAR>
<PAGESIZE>-1</PAGESIZE>
<RESTRICTION>?y rdf:type lemon:LexicalEntry
 </RESTRICTION>
<RESTRICTION>FILTER langMatches( lang(?v0), "en" )
 </RESTRICTION>
<PROPERTY>rdfs:label AS lowercase->nolang
RENAME label </PROPERTY>
</TARGET>
<METRIC>trigrams(x.label,y.label)</METRIC>
```

We evaluated the quality of the links generated by manually checking 100 randomly selected links from each of the three languages. The manual check was carried out by the two authors. A link was set to be correct if both authors agreed on it being correct. Overall, the linking achieve a precision of 100% for the English version, 96% for the French and 87% for the German. The error in the French links were due homonymy errors. For example, “Est” (engl. East) was linked to “est” (engl. to be) in some cases. Similarly in the German, “Stütze” (engl. support) was linked to “stütze” (engl. imperative singular form the verb “to support”). In the next version of the dataset, we will add context-based disambiguation techniques to improve the quality of the links. Especially, we will consider the type of the expression to link while carrying out the linking to ensure that verbs cannot be matched with nouns for example. Still, the accuracies we achieve in these three languages are sufficient to make the dataset useful for NLP applications. The recall could not be computed manually. While these values are satisfactory, they can be improved further by devising a disambiguation scheme based on the context

within which the words occurred. To achieve this goal, we aim to combine the results of LIMES with the AGDISTIS disambiguation framework[17] in future work.

27.6 USE CASES

The availability of a multilingual parallel corpus in RDF promises to facilitate a large number of NLP applications. In this section, we outline selected application scenarios and use cases for our dataset.

DATA RETRIEVAL. The Quran contains a significant number of instances of places, people and events. Thus, multilingual sentences concerning such information can be easily retrieved from our dataset, for example for the purpose of training NLP tools. Moreover, the aligned multilingual representation allows searching for the same entity across different languages. For example, Listing 27.2 shows a SPARQL query which allows retrieving Arabic, English and German translations of verses which contain *"Moses"*.

Listing 27.2: Verses that contains moses in Arabic, English and German

```
SELECT DISTINCT ?chapterIndex ?verseIndex
       ?verseTextAr ?verseTextEn ?verseTextGr
WHERE{
  ?word rdfs:label "Moses"@en;
        dcterms:isPartOf ?verse.
  ?verse a qvoc:Verse;
         skos:prefLabel ?verseTextAr;
         qvoc:verseIndex ?verseIndex;
         dcterms:isPartOf ?chapter;
         rdfs:label ?verseTextEn;
         rdfs:label ?verseTextGr.
  FILTER ( lang(?verseTextEn) = "en" &&
           lang(?verseTextGr) = "de")
  ?chapter qvoc:chapterIndex ?chapterIndex.
}
```

ARABIC LINGUISTICS. The RDF representation of Arabic morphology and syntax promises to facilitate the retrieval of relevant sub-corpora for researchers in linguistics. For example, Listing 27.3 provides an example of a SPARQL query witch retrieves all Arabic prepositions as well as an example statement for each of them.

Listing 27.3: List all the Arabic prepositions and show an example statement for each of them

```
SELECT ?preposition
  ( sql:SAMPLE ( ?verseTextAr ) AS ?example )
WHERE{
  ?s gold:PartOfSpeechProperty olia-ar:Preposition;
     skos:prefLabel ?preposition;
```

17 http://github.com/AKSW/AGDISTIS

```
    dcterms:isPartOf ?verse.
  ?verse a qvoc:Verse;
         skos:prefLabel ?verseTextAr.
}GROUP BY ?preposition
```

Another example is provided by Listing 27.4, which shows a list of different part-of-speech variations of one Arabic root of the word read *"ktb"* (engl. "write"); note that in this example we use the Arabic root *"ktb"* written in The Buckwalter transliteration.

Listing 27.4: List of different part of speech variations of one Arabic root of the word read *"ktb"*.

```
SELECT DISTINCT ?wordText ?pos
WHERE{
  ?wordPart a qvoc:LexicalItem ;
            gold:Root "ktb";
            gold:PartOfSpeechProperty ?pos;
            dcterms:isPartOf ?word.
  ?word a qvoc:Word;
        skos:prefLabel ?wordText.
}
```

INTEROPERABILITY USING NIF. Using the interoperability capabilities provided by NIF, it is easy to query all occurrences of a certain text segment without using the verse, chapter, word, or lexical item indexes. For instance, Listing 27.5 lists all the occurrences of *"Moses"* with no need to have an extra index.

Listing 27.5: List of all occurrences of "Moses" using NIF

```
SELECT ?textSegment ?verseText {
  ?s str:occursIn ?verse;
     str:isString ?verseText.
  ?textSegment str:referenceContext ?s;
               str:anchorOf "Moses"@de.
}
```

INFORMATION AGGREGATION. The interlinking of the Quran dataset with other RDF data sources provides a considerable amount of added value to the dataset. For example, the interlinking with Wiktionary can be used as in Listing 27.6 to get the different senses for each of the English words contained in the first verse of the first chapter *"qrn:quran1-1"*.

Listing 27.6: List of all senses of all English words of the first verse of the first chapter "qrn:quran1-1"

```
SELECT DISTINCT ?wordTextEn ?sense
FROM <http://thedatahub.org/dataset/semanticquran>
FROM <http://en.wiktionary.dbpedia.org>
WHERE {
  ?word a qvoc:Word .
```

```
  ?word rdfs:label ?wordTextEn .
  ?word dcterms:language lexvo:eng .
  ?word dcterms:isPartOf qrn:quran1-1 .
  ?word owl:sameAs ?wiktionaryWord .
  FILTER(lang(?wordTextEn)="en")
  ?wiktionaryWord lemon:sense ?sense .
}
```

27.7 CONCLUSION AND FUTURE WORK

In this work, we presented the Semantic Quran, an integrated parallel RDF dataset in 42 languages. This multilingual dataset aims to increase the availability of multilingual data in LOD and to further the development of NLP tools for languages that are still under represented, if not absent, from the LOD cloud. Thanks to its RDF representation, our dataset ensures a high degree of interoperability with other datasets. For example, it provides 26735 links overall to Wiktionary and DBpedia. As demonstrated by our use cases, the dataset and the links it contains promise to facilitate research on multilingual applications. Moreover, the availability of such a large number of languages in the dataset provides opportunities for linking across the monolingual datasets on the LOD Cloud and thus perform various types of large-scale analyses.

To improve the ease of access to our dataset, we aim to extend the TBSL framework (Unger et al., 2012) to allow even lay users to gather sensible information from the dataset. Moreover, we aim to provide links to the upcoming versions of Wiktionary. Additionally, we will link the Semantic Quran dataset with many of the publicly available multilingual *Wordnets*. We already provided NIF for the five languages Arabic, English, French, German and Russian. We will extend the NIF content of the dataset to the remaining 38 languages.

Part VI

CONCLUSIONS

This section summarizes our findings and presents future work in the area of link discovery. In Chapter 28, we focus especially on summarizing the results presented mainly in Part II, Part III and Part IV. The future work discussed in Chapter 29 presents some of the future research directions we aim to investigate.

28 SUMMARY AND CONCLUSIONS

The main aim of this work was to present solutions to the two challenges presented in Chapter 1. In Part II, we addressed the *time-complexity challenge*. Our first approach, presented in Chapter 3, used the triangle inequality to portion space by means of exemplars. This technique showed a significant improvement over the state of the art but did not provide any theoretical guarantees with respect to the type of improvements it could achieve. A step toward a solution with theoretical guarantees was presented in Chapter 4, where we presented HYPPO, an approach with a guaranteed relative reduction ratio (RRR). While HYPPO clearly outperformed the state of the art, its RRR was not guaranteed to be 1. Rather, we showed that its RRR grows with the volume of the hypersphere of diameter 1. We thus pursued our research and developed the first reduction-ratio optimal approach for link discovery in spaces with Minkowski distances, dubbed $\mathcal{HR}^3$. The approach, which is presented in Chapter 5, was the basis for ORCHID (Chapter 6), which is also reduction-ratio-optimal but can be applied in orthodromic spaces. The approaches we developed showed clearly that several orders of magnitude in runtime can be gained by using the characteristics of similarity measures used in link discovery to predict comparisons that need not be carried out. While this can be a complex endeavor, our approaches seem to be easily portable to other types of affine spaces. For example, the sequence of filters presented in Chapter 7 has been ported to the Jaro-Winkler similarity measure in (Dreßler and Ngonga Ngomo, 2014).

In Part II, we also investigated planning as another means to improve the computation of complex link specifications. We developed the first planner for link discovery and dubbed it HELIOS (see Chapter 8). With this approach, we showed that improving the runtime of link discovery can also be carried out by optimizing the sequence in which sub-specifications are executed. Another avenue of research that we focused on was the parallel execution of link specifications within various hardware paradigms (see Chapter 9 and Chapter 10). Our insights (which were crowned with a best research paper award at ESWC 2013) led us to suggest that an intelligent combination of several parallel computation paradigms is the best path toward improving the runtime of link discovery even further.

Part III focused on the second challenge, i.e., the *accuracy challenge*. With RAVEN, we proposed the first active learning approach for link discovery (see Chapter 11). We also showed that this paradigm is amenable to other types of learning. In particular, we showed in Chapter 12 that genetic programming can also be used in combination with active learning. This particular approach to learning was further extended by a novel approach toward finding the most informative positive and negative examples. With COALA (see Chapter 13), we showed that using intra- and inter-class similarity allows for a better detection of informative links and thus a better selection of oracle queries while learning. Another type of learning was presented in Chapter 14, where we considered deterministic (EUCLID) and non-deterministic (EAGLE) unsupervised learning for discovering link specifications. Overall, our results suggest that active learning supersedes batch learning, as the correct choice of examples leads classifiers to converge faster. Adequate approaches for choosing

the right examples also have a positive effect on the convergence of the underlying machine learning approaches. While genetic programming was shown to be viable for link discovery in many cases, deterministic approaches also perform well. In particular, the comparison of classification techniques presented in Chapter 16 suggests that the state-of-the-art approaches are very close in performance when it comes to link discovery. Still, our results suggest that dedicated machine learning techniques designed for link discovery can still go beyond the performance of current link discovery approaches.

The COLIBRI approach described in Chapter 15 marked a departure from the classical learning approach using two knowledge bases. Here, we studied how using several knowledge bases at the same time can be used to improve the quality of the data while linking. We showed that this paradigm can yield significant improvements in comparison to a classical linking approach with two knowledge bases. Moreover, the addition of suggestions of quality improvements through linking extended the idea of link discovery to a new dimension. ROCKER (see Chapter 17) also tapped into linking combined with quality improvements and showed how keys for linking can be computed efficiently. With keys, errors in RDF data can be detected effectively and presented to the end user (Soru et al., 2015a).[1]

Part IV brought Part II and Part III together into a single framework, the LIMES framework (see Chapter 18).[2] In addition to providing details pertaining to how to use the framework (Chapter 18 and Chapter 19) and manage links through a repository (Chapter 20), Part IV presented applications of LIMES in Part V. First, we showed how the framework can be used for creating benchmarks for triple stores (Chapter 21 and Chapter 22). Here, the runtime efficiency of LIMES was of particular importance when computing the similarity of the millions of queries found in query logs. This particular application was crowned with a best research paper award at ISWC 2011. Another usage of LIMES was in the generation of data for question answering engines. In Chapter 23, we showed how question answering can profit from the explication of relations between RDF resources from different knowledge bases. Chapter 24 presented a further application of the LIMES algorithms, where we computed the similarity of natural-language patterns that potentially express RDF predicates to improve open knowledge extraction. The last three chapters of the part, Chapter 25, Chapter 26 and Chapter 27, showed LIMES being used for the purpose for which it was created initially, i.e., linking knowledge bases across the Web of Data. With Part IV and Part V of this work, we showed the applicability of the solutions we developed for link discovery. In particular, we displayed that the time-efficient and accurate algorithms we developed can be used beyond the area of link discovery and can thus be regarded as an enabler for time-efficient RDF-driven applications where similarity computations are needed.

Overall, our work shows how theoretical considerations pertaining to the complexity and accuracy of link discovery led to a practical implementation (the LIMES framework) which was used in diverse applications. While this thesis shows that a significant amount of progress was achieved over the last years (for example, see the improvements from RAVEN to COALA), it also suggests that a considerable amount of work still needs to be done to achieve highly scalable and accurate link discovery systems, especially within the upcoming era of Big Linked Data.

1 See http://rocker.aksw.org/. Accessed August 11, 2015.
2 http://limes.sf.net. Accessed August 11, 2015

29 FUTURE WORK

Our long-term vision for link discovery is that of time-efficient, self-configuring and self-learning systems. While we presented an extensive set of approaches that can build the foundations for such a vision in the previous chapters, there remains a significant amount of work to undertake so as to achieve this vision. The aim of this chapter is to provide an overview of short-term goals that we aim to reach so as to make this long-term vision a reality.

29.1 AUTOMATIC OPTIMIZATION OF MEASURES

One observation that we made early on is that many of the time-efficient approaches for similarity computation found in the current literature (Xiao et al., 2008; Li et al., 2011; Feng et al., 2012; Soru and Ngonga Ngomo, 2013; Dreßler and Ngonga Ngomo, 2014) rely on a similar sequence of filters (including a length-based filter and a character-based filter). While several modern approaches rely on further or different filters (Georgala et al., 2016), one interesting research direction could consist of devising automated means to devise the correct formulae for these two basic filters. A possible approach toward this goal would be to rely on refinement operators that enable the derivation of formulae based on arithmetical operators combined with a basic set of symbols such as string lengths, the similarity threshold and numerical constants. While this automated approach toward deriving thresholds for measures could not prove that the equations it generates are correct, it could be used as a tool to support researchers during the development of rapid execution algorithms for certain measures.

29.2 LEARNING LINK SPECIFICATIONS

A number of improvements can also be made to increase the accuracy of link discovery. One promising research area is the extension of approaches for the selection of most informative examples within the active learning paradigm. With (Ngonga Ngomo et al., 2013), we showed that the convergence of genetic programming toward good solutions can be improved simply by clustering a selection of examples or using information transfer. Incremental clustering (Georgala et al., 2014) and similar techniques seem to promise even better results when combined with the aforementioned approaches. Other extensions of committee-based approaches (Settles, 2012) will also be considered in the future. In addition, the promising results of semi-supervised learning and boosting (Kejriwal and Miranker, 2015) suggest that other learning paradigms also allow for reducing the number of examples used for learning while achieving high accuracy.

A second strand of research pertains to multi-relational learning. Approaches based on tensor factorization (Nickel et al., 2012) have been shown to scale up to millions of triples and to return highly accurate results for certain properties. The main drawback of these approaches is that they require a large number of training examples to converge toward good solutions. Combining multi-relational

learning with active learning or weakly supervised learning has the potential of being the basis for the next generation of link discovery approaches. This should hold especially when these approaches are combined with cross-knowledge-base learning (Ngonga Ngomo et al., 2014) and transfer learning (Ngonga Ngomo et al., 2013), where more than 2 knowledge bases are used at the same time.

29.3 PLANNING AND RESOURCE MANAGEMENT

Planning remains a vastly unexplored part of link discovery. While HELIOS (Ngonga Ngomo, 2014) presents the formal foundations for planning during the execution of specifications, our approach does not consider some dimensions that might lead to better planning. First, a dynamic planning routine could improve the approximation of the size of the mappings as well as of the runtimes of the sub-specifications. In addition, the functions used in the algorithm to approximate the runtime and mapping size of a specification are solely based on linear features. Using more complex non-linear functions could potentially lead to better approximations and thus runtimes. Moreover, taking the parallel execution of portions of specifications into consideration could improve the runtime even further. Finally, an ensemble learning method could be used to merge the results of different planning approaches.

With planning also come novel areas of research such as resource management for link discovery (Hassan et al., 2015; Ngonga Ngomo and Hassan, 2016). With the increase of the velocity of the generation of Linked Data comes the need to link data in a continuous manner. For example, application areas such as industry 4.0 (see research projects such as SAKE[1]) and geo-spatial data management (see GeoKnow project[2]) require the continuous integration of RDF data streams. To achieve this goal, conditions such as achieving 100% recall might have to be dropped for other conditions such as computing as many links as possible within a the buffering time of the streams. Achieving linking within such conditions requires revisiting planning and execution algorithms and deriving a new generation of approaches with guaranteed recall and/or guaranteed runtime. Moreover, priorities might be assigned to certain linking tasks within time-critical applications, leading to novel approaches pertaining to space and time management for link discovery–problems which are commonly NP-complete.

29.4 AUTOMATION OF THE LINKED DATA LIFE CYCLE

A large number of the approaches developed within this work had their focus on improving link discovery. However, many of the ideas presented herein can be extended to profit the whole of the Linked Data lifecycle (see Figure 29.1).

- During the *extraction phase*, our time-efficient algorithms can be used to detect similar labels that most probably stand for the same resource (Gerber et al., 2013).
- Our prefix-based algorithms can also play a role during the *storage phase*. Previous works have shown that graph-based data compression can improve the storage of RDF data. By using our algorithms, the detection of the graph

1 http://sake-projekt.de
2 http://geoknow.eu

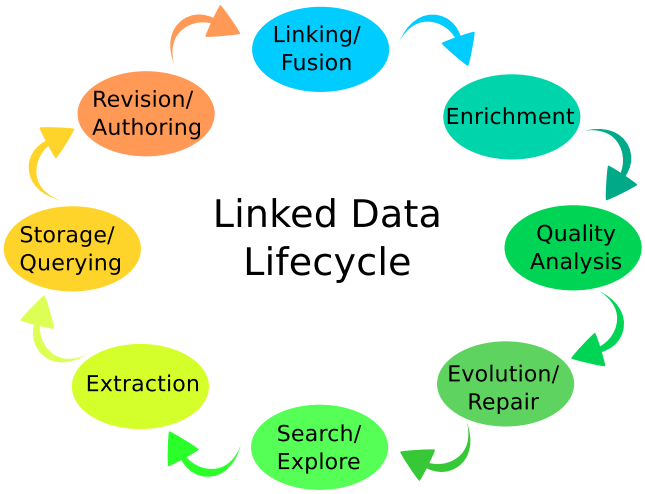


Figure 29.1: The Linked Data lifecycle (Auer et al., 2013b)

patterns that point to portions of the graph that should be folded to one node during the compression could be improved (Pan et al., 2014).

- *Authoring* can also profit from the results of linking as the interfaces can be adapted to display `owl:sameAs`-resources in a way that makes their common identity clear.
- The influence of link discovery on *data enrichment* is also clear, as the a-priori processing of knowledge bases through linking provides more formal background knowledge upon which enrichment approaches can learn (Lehmann et al., 2011; Ngonga Ngomo et al., 2014; Sherif et al., 2015).
- Richer knowledge bases also allow for a better assessment (as well as for an improvement) of the *quality of linked datasets* as they contain explicit relations across resources that were not available before the enrichment. Especially automated frameworks, such as RDFUnit (Kontokostas et al., 2014), profit from this explicit knowledge as they rely on SPARQL templates for quality checking that most commonly count incidences to determine quality.
- Supporting the *evolution of linked datasets* requires supporting the efficient computation of novel links as well as the deletion of invalid links. Achieving this process efficiently and effectively is of central importance in critical applications such as error prediction of sensor streams, biomedical applications, finances and many more.
- Finally, the *exploration of Linked Data* requires explicit links to enable a rapid navigation between related entities in a similar way to how hyperlinks between Web pages are needed for navigating across the Document Web.

Within the next years, we thus will aim to port our algorithms to other steps of the Linked Data lifecycle. With these efforts, we aim to improve the (semi-)automation of more steps of the Linked Data lifecycle. Our overall aim will be to make Linked Data management usable for lay users even when faced with highly complex data and data processing pipelines.

30 ZUSAMMENFASSUNG

In diesem Kapitel wird diese Habilitationsschrift auf Deutsch zusammengefasst. Dazu wird ein Überblick über die erarbeiteten Verfahren sowie die erzielten Ergebnisse gegeben. Technische Details können den entsprechenden Kapiteln entnommen werden. Bei den Übertragungen der Fachbegriffe handelt es sich um die Übersetzungen des Verfassers.

30.1 MOTIVATION

Das mittlerweile 3 Jahrzehnte alte World Wide Web hat sich zu einem Kompendium aus mehreren Exabytes an Daten entwickelt. Es wird nun von Milliarden von Menschen für eine Vielzahl von diversen Aktivitäten genutzt. Eine der wichtigsten Anwendungen des World Wide Webs ist die Suche nach Informationen. Erfolgreiche Portale wie Wikipedia[1] stellen gebündelte Informationen zu einer Vielzahl von Entitäten zur Verfügung. Wikipedia enthält zum Beispiel alle Informationen, die zur Beantwortung der Frage

> Welche argentinischen Fußballspieler haben in der Premier League gespielt?

benötigt werden. Jedoch ist die Beantwortung solcher Fragen mit Hilfe von klassischen Suchmaschinen ein schwieriges Unterfangen. Grund dafür ist die Verteilung der zur Beantwortung der Frage benötigten Informationen über mehrere Webseiten. Erheblich anspruchsvoller wird die Problematik, wenn Fragen wie

> Was sind die Nebenwirkungen von Medikamenten für mit dem Gen FOXP2 assoziierte Krankheiten?

zu beantworten sind. Hier sind die benötigten Informationen über mehrere Datenquellen hinweg verteilt, wie z. B. DailyMed,[2] Drugbank[3] und Sider.[4]

Das semantische Web (Übersetzung des Verfassers, engl.: Semantic Web) zielt darauf ab, den effizienten Zugriff auf Informationen im Web zu verbessern. Der Grundtenor hinter der vorgeschlagenen Lösung ist die Verbesserung der Verarbeitung von Wissen aus dem Web durch Maschinen. Berners-Lee et al. (2001) beschreiben ein semantisches Web, wo Software-Agenten Informationen über mehrere Webseiten hinweg sammeln können. Die explizite Semantik der Informationen vereinfacht und beschleunigt die automatisierte Zusammenführung dieser Informationen. Dadurch wird die Umsetzung komplexer informationsgetriebener Prozesse ermöglicht. Zu Beispiele solcher Prozesse gehören unter anderem die Beantwortung der oben genannten Fragen, die Synchronisierung von Kalendereinträgen sowie die Entdeckung von relevanten domänenspezifischen Fakten in Rahmen von wissenschaftlichen Anwendungen.

Eine Vielzahl von formalen Sprachen wurde über die letzten Jahre entwickelt, um diese Vision zu verwirklichen. Die Sprachen RDF[5] (Resource Descrip-

1 http://wikipedia.org
2 http://dailymed.nlm.nih.gov/dailymed/
3 http://www.drugbank.ca/
4 http://sideeffects.embl.de/
5 http://www.w3.org/RDF/

tion Framework, dt.: Rahmenwerk zur Beschreibung von Ressourcen, Übersetzung des Verfassers), RDFS[6] (Resource Description Framework Schema, dt.: Rahmenwerk zur Beschreibung von Ressourcenschemata, Übersetzung des Verfassers), OWL[7] (Web Ontology Language, dt.: Web-Ontologie Sprache, Übersetzung des Verfassers) und SPARQL[8] (SPARQL Protocol and RDF Query Language, dt.: SPARQL Protokoll und RDF Anfragesprache, Übersetzung des Verfassers)[9] gelten mittlerweile als die Standardsprachen des semantischen Webs. Mit Hilfe dieser Sprachen wurde der Wegbereiter für das semantische Web, das Linked Data Web (dt.: Web verknüpter Daten, Übersetzung des Verfassers), ins Leben gerufen. Er umfasst nun circa 10.000 Wissensbasen und 150 Milliarden Fakten.[10] Wie das Web der Dokumente (engl.: Document Web, Übersetzung des Verfassers) unterliegt das Linked Data Web einer Menge von Prinzipien:[11]

> **Prinzip 1**: Verwende URIs, um Dinge zu kennzeichnen.
>
> **Prinzip 2**: Verwende HTTP URIs, damit diese Dinge gefunden werden können.
>
> **Prinzip 3**: Stelle nützliche Informationen zu einer URI zur Verfügung, wenn nach ihr gesucht wird.
>
> **Prinzip 4**: Stelle Verknüpfungen zu anderen URIs zur Verfügung, damit mehr Dinge entdeckt werden können.

Eine Plethora von Werkzeugen zur Transformation von nicht-RDF-Daten nach RDF wurde in den letzten Jahren entwickelt. Ansätze wie Cool URIs[12] wurden im Rahmen dieser Anstrengungen erarbeitet. Frameworks wie D2R (Eisenberg and Kanza, 2012),[13] SPARQLIFY (Stadler et al., 2015)[14] und Virtuoso[15] gehören zum gegenwärtigen Stand der Technik zur Umsetzung des zweiten und des dritten Prinzips. Die vorliegende Arbeit setzt sich mit dem vierten Prinzip von Linked Data auseinander, das unter dem Namen Link Discovery (engl.: Entdeckung von Verknüpfungen, Übersetzung des Verfassers) bekannt ist.

Sei S eine Menge von RDF Ressourcen. Das Ziel der Arbeit ist die Erarbeitung von Verfahren, welche die Verknüpfung von Ressourcen aus S mit anderen Mengen von Ressourcen ermöglichen. Dieses Prinzip ist von äußerster Wichtigkeit, um ein semantisches Web umzusetzen, da es das Fundament für die Migration von Datensilos hin zu verlinkten und interoperablen Wissensbasen darstellt. Dadurch werden auch Anwendungen wie die Beantwortung von Fragen über verteilte Daten (Bhagdev et al., 2008; Lopez et al., 2009; Shekarpour et al., 2013), großskalige Inferenz (Urbani et al., 2010; McCusker and McGuinness, 2010), Datenintegration (Ma et al., 2009; Ben-David et al., 2010) und föderierte Anfragen (Schmidt et al., 2011; Saleem and Ngonga Ngomo, 2014) im semantischen Web realisierbar.

---

6 http://www.w3.org/TR/2014/REC-rdf-schema-20140225/
7 http://www.w3.org/standards/techs/owl#w3c_all
8 http://www.w3.org/TR/sparql11-query/
9 SPARQL ist ein rekursives Akronym.
10 http://stats.lod2.eu/, Stand vom 19. Juli 2016.
11 http://www.w3.org/DesignIssues/LinkedData.html, Stand vom 19. Juli 2016, übersetzt aus dem Englischen durch den Verfasser.
12 http://www.w3.org/TR/cooluris/
13 http://d2rq.org/d2r-server
14 http://aksw.org/Projects/Sparqlify.html
15 http://virtuoso.openlinksw.com/

Eine Vielfalt von Lösungen können zur Verknüpfung von Wissensbasen eingesetzt werden. Ein manueller Ansatz wäre eine Möglichkeit. Die folgende Berechnung zeigt jedoch, dass dieser Ansatz zum Scheitern verurteilt ist: Die Wissensbasis DBpedia[16] (aus Wikipedia abgeleitet) allein beschreibt mehr als 3 Millionen Dinge. Die Wissenbasis LinkedGeoData[17] (aus OpenStreetMap) enthält mehr als 10 Millionen Ressourcen. Die manuelle Erarbeitung von Verknüpfungen zwischen diesen Wissensbasen würde des Vergleichs von mehr als 3×10^{13} Paaren von Entitäten bedürfen. Übernähmen 1 Milliarde Menschen diese Aufgabe und dauerte das Überprüfen eines Paars nur 1 Minute, dann wäre von einer Dauer von mehr als 10 Wochen auszugehen.[18] Im Linked Data Web sind ca. 50 Millionen Paare von Wissensbasen zu finden. Es wird deutlich, dass (semi-)automatisch Ansätze unabdingbar sind, um Wissensbasen miteinander zu verknüpfen.

30.2 ZIEL DER ARBEIT

Ziel dieser Arbeit ist die Erarbeitung von effizienten (semi-)automatischen Verfahren zur Verknüpfung von Wissensbasen. Eine Vielzahl von Lösungsklassen können zu diesem Zweck eingesetzt werden. In dieser Arbeit werden ausschließlich deklarative Ansätze erörtert. Die formale Formulierung des Link-Discovery Problems hinter diesen Ansätzen lautet folgendermaßen:

> Gegeben seien zwei Mengen von Ressourcen S und T sowie eine Relation R. Finde die Menge $M = \{(s,t) \in S \times T : R(s,t)\}$.

Deklarative Ansätze gehen davon aus, dass das direkte Errechnen von M in vielen Fällen nur schwer möglich ist oder eines nicht vertretbaren Aufwands bedarf (siehe Beispiel unter Abschnitt 30.1). Diese Ansätze zielen daher darauf ab, eine Ähnlichkeitsfunktion $\sigma : S \times T \rightarrow [0,1]$ sowie einen Schwellwert $\theta \in [0,1]$ zu finden, welche zur Folge haben, dass die Menge $M' = \{(s,t) \in S \times T : \sigma(s,t) \geqslant \theta\}$ die Menge M approximiert. Äquivalente auf Distanzen basierende Formulierungen kommen ebenfalls zur Anwendung. Zwei Herausforderungen gehen mit dieser Modellierung des Problems einher: (a) Effizienz sowie (b) Genauigkeit und Vollständigkeit.

30.3 EFFIZIENZ

Die Herausforderung hinsichtlich der Effizienz ist eine direkte Konsequenz der Formulierung des Link Discovery Problems unter dem deklarativen Paradigma. Naive Ansätze werten $\sigma(s,t)$ für alle Paare aus $S \times T$ aus, um M' zu berechnen. Während solche Ansätze für kleine S und kleine T eingesetzt werden können, besteht die Möglichkeit, dass die Berechnungszeiten hunderte von Jahren überschreiten, wenn S und T große Wissensbasen sind. Zum Beispiel würde der Vergleich aller Ressourcen aus DBpedia mit allen Ressourcen aus LinkedGeoData in etwa 951 Jahre andauern, wenn davon ausgegangen wird, dass eine Ähnlichkeitsberechnung 1 ms dauert.[19]

In dieser Arbeit wird eine Vielzahl von Verfahren vorgestellt, die darauf abzielen, zeiteffizienter als naive Ansätze zu sein. In Kapitel 3 wird der LIMES Ansatz

16 http://dbpedia.org
17 http://linkedgeodata.org
18 Angenommen werden 8h Arbeit pro Tag, 7 Tage/Woche.
19 Angenommen wird eine rein sequentielle Berechnung.

vorgestellt (Ngonga Ngomo and Auer, 2011). LIMES geht von der Formulierung des deklarativen Link Discovery Problems mittels Distanzen aus. Hier wird die Menge M durch $M' = \{(s,t) \in S \times T : \delta(s,t) \leqslant \tau\}$ approximiert, wo $\delta : S \times T \rightarrow \mathbb{R}^+$ ein Maß für die Unähnlichkeit zweier Ressourcen und $\tau \in \mathbb{R}^+$ der entsprechende Schwellwert ist. Die Grundidee hinter diesem Ansatz ist wie folgt: Falls δ eine Metrik (im mathematischen Sinne) ist, dann müssen nicht alle (s,t)-Paare miteinander verglichen werden. Stattdessen basiert der Ansatz auf einer Raumteilung, die es ermöglicht, Distanzen zu approximieren bevor sie berechnet werden. Die Raumteilung wird durch die Kombination von Exemplaren mit der Cauchy-Schwarz-Ungleichung für Distanzen umgesetzt. Die Evaluation des Verfahrens zeigt, dass es dem damaligen Stand der Technik signifikant überlegen war.

Die Verbesserung der Effizienz von Verfahren in affinen Räumen ist auch der Fokus von Kapitel 4. Hier wird die Bedingung $\delta(s,t) \leqslant \tau$ näher untersucht und als Beschreibung einer Hyperkugel H um $s \in S$ interpretiert (Ngonga Ngomo, 2011, 2012b). Ein Approximationsverfahren für Hyperkugeln H mittels Hyperwürfel W wird erarbeitet. Es wird gezeigt, dass die sich aus diesem Verfahren ergebende Approximation $|W|/|H|$ gegen ein Minimum strebt.[20] Dieser Minimalwert ist jedoch nicht 1 und wächst mit dem Verhältnis der Volumen von Einheits-Hyperwürfeln zu Einheits-Hyperkugeln. Die Evaluation zeigt jedoch deutlich, dass das erarbeitete Verfahren auch hier dem Stand der Technik überlegen ist.

Die Erarbeitung einer optimalen Lösung für das oben genannte Approximations-Problem ist das Ziel von Kapitel 5 (Ngonga Ngomo, 2012a). Die vorgestellte Methode kombiniert die Lösung aus dem vorangegangenen Kapitel mit Minkowski-Metriken sowie mit einem numerischen Index. Dieser Index ermöglicht es, Teile von W bei der Approximation von H zu ignorieren ohne Elemente aus M' zu verlieren. In diesem Kapitel wird bewiesen, dass die erzielte Qualität der Approximation in dem Sinne optimal ist, dass sie gegen den bestmöglichen Wert 1 strebt, wenn die Granularität der Approximation gegen ∞ strebt. Damit wurde der erste optimale Ansatz für Link Discovery in affinen Räumen mit Minkowski-Metriken entwickelt. Die Evaluation des Ansatzes zeigt, dass die praktische Implementierung des Ansatzes die Versprechen der theoretischen Erarbeitung hält und den damaligen Stand der Technik übertrifft.

Eine Frage, die sich aus den Untersuchungen in Kapitel 5 ergibt, betrifft die Übertragbarkeit der erarbeiteten Lösungen auf Räume mit nicht-Minkowski-Metriken. Räume mit orthodromischen Distanzen werden in Kapitel 6 untersucht. Es wird gezeigt, dass eine einfache Modifizierung des Indexierungsverfahrens zu einem optimalen Approximationsverfahren auch in solchen Räumen führt (Ngonga Ngomo, 2013). Es wird insbesondere aufgezeigt, wie orthodromische Distanzen auf Ebenen projiziert werden können. Das sich daraus ergebende Verfahren wird mit dem damaligen Stand der Technik verglichen und für zeiteffizienter als der Stand der Technik befunden.

Die nachfolgenden Kapitel untersuchen weitere Ansätze zur Verbesserung der Laufzeit von Link Discovery. In Kapitel 7 wird zunächst untersucht, ob die Skalierbarkeit von domänenspezifischen Metriken verbessert werden kann (Soru and Ngonga Ngomo, 2013). Dazu wird die gewichtete Levenshtein-Metrik analysiert. Es wird gezeigt, dass es die mathematischen Eigenschaften von gewichteten Metriken ermöglichen, Paare von Ressourcen aufgrund der Komposition der zu vergleichenden representativen Zeichenketten (z. B., Werte von Prädikaten) zu verwerfen ohne

20 $|H|$ resp. $|W|$ stehen für das Volumina von H resp. W.

diese miteinander zu vergleichen. Dadurch kann die für die Berechnung von M' notwendige Gesamtlaufzeit wesentlich verkürzt werden. Die entwickelte Menge von Filtern wird evaluiert und zeigt eine gute Skalierbarkeit.

In Kapitel 8 wird ein weiterer neuartiger Ansatz verfolgt (Ngonga Ngomo, 2014), nämlich der Einsatz von Planungsalgorithmen. Die zu berechnenden Ähnlichkeitsfunktionen σ sind meist komplexer Natur, d. h., sie bestehen oft aus einer algebraischen Zusammenführung von atomaren Metriken. Das Kapitel beschäftigt sich mit der globalen Optimierung der Ausführungen derartiger Ähnlichkeitsfunktionen. Die Existenz von Optimierungsverfahren für atomare Metriken wird vorausgesetzt. In dem Kapitel wird zunächst gezeigt, wie die Laufzeit sowie die Größe der Ergebnismenge von atomaren sowie von komplexen Ähnlichkeitsfunktionen approximiert werden können. Dazu wird eine Kombination aus Stichproben, linearer Regression und probabilistischen Modellen verwendet. Komplexe Metriken können als Bäume dargestellt werden. Der zweite Schritt des Ansatzes umfasst die Verarbeitung der Ähnlichkeitsfunktionen von den Blättern hin zur Wurzel des Baums. Hier werden die zur Verknüpfung von atomaren Metriken verwendeten Bool'schen Operatoren auf Pläne abgebildet, deren Laufzeit approximiert wird. Die approximierten Laufzeiten werden genutzt, um den (der Approximation nach) schnellsten Ausführungsplan zu finden, welcher anschließend ausgeführt wird. Der HELIOS Ansatz ist der erste Planner für Link Discovery und die Evaluation des Ansatzes zeigt, dass der Planner die Laufzeit von Link Discovery Prozesse deutlich verbessert. Insbesondere kann der Planner auch bei kleinen (Anzahl der Knoten im Baum $\leqslant 3$) Metriken eingesetzt werden und ist in der Lage, einige Berechnungen mehrere Größenordnungen schneller als ein kanonischer Ansatz auszuführen.

Untersucht werden auch verschiedene parallele Hardware-Architekturen für die Berechnung von Links zwischen Wissensbasen. In Kapitel 9 wird eine auf Map-Reduce basierende Ausführungsumgebung für den in Kapitel 3 vorgestellten Algorithmus erarbeitet (Hillner and Ngonga Ngomo, 2011). Das Kapitel beschreibt die Mapping- sowie die Reduktionsschritte, welche zur Ausführung von LIMES benötigt werden. Evaluiert wird die Skalierbarkeit der Implementierung mittels echter Daten aus DBpedia. Es wird gezeigt, dass die parallele Version von LIMES wesentlich zeiteffizienter ist als der damalige Stand der Technik.

In Kapitel 10 werden mögliche Hardware-Architekturen für Link Discovery verglichen. Zusätzlich zum Map-Reduce Paradigma in der Cloud werden auch GPUs sowie lokale CPU-Threads eingesetzt (Ngonga Ngomo et al., 2013). Als Metrik wird die Euklidische Distanz in Kombination mit dem Verfahren aus Kapitel 5 gewählt. Die Ergebnisse zeigen, dass lokale Ressourcen (insbesondere der hybride Einsatz von CPUs und GPUs) enfternen Ressourcen überlegen sind, besonders wenn S und T bis zu ca. 10^6 Ressourcen enthalten und weniger als 20 entfernte Verarbeitungsknoten genutzt werden. Der Grund dafür sind die Kosten des Transports von Daten von lokalen zu entfernten Rechenressourcen. Dementsprechend wird die Empfehlung ausgesprochen, in den meisten Fällen lokale Ressourcen zu verwenden. Diese neue Erkenntnis wurde mit dem Best Research Paper Award der Extended Semantic Web Conference 2013 gekürt.

30.4 LERNEN VON SPEZIFIKATIONEN

Die zweite Herausforderung des Link Discovery Problems ergibt sich ebenso aus der Spezifikation der Menge M'. Hier liegt die Herausforderung darin, eine Ähnlichkeitsfunktion σ sowie einen entsprechende Schwellwert θ zu finden, so dass M' eine hohe Genauigkeit (engl.: precision, Übersetung des Verfassers) sowie eine hohe Vollständigkeit (engl.: recall, Übersetzung des Verfassers) bezüglich M erzielt. Wie sowohl aus voragegangenen Abschnitt als auch aus Abbildung 1.2 hervorgeht, sind die Funktionen σ oft komplexe Funktionen, die sich aus der Verknüpfung einer Vielzahl von atomaren Funktionen mittels Boolescher Operatoren ergeben. Die Bestimmung guter Kombinationen von atomaren Funktionen ist dementsprechend ein höchst komplexes Problem, welches in dieser Arbeit mittels verschiedener maschineller Lernverfahren angegangen wird.

Kapitel 11 setzt sich mit dem Lernen von Ähnlichkeitsfunktionen auseinander. Das Link Discovery Problem wird zunächst auf ein äquivalentes binäres Klassifikationsproblem abgebildet, bei dem alle Paare (s,t) mit $\sigma(s,t) \geqslant \theta$ zur Klasse $+1$ gehören (Ngonga Ngomo et al., 2011). Die Paare für die $\sigma(s,t) < \theta$ gilt, werden der Klasse -1 zugeordnet. Das Problem besteht nun darin, eine richtige Funktion σ sowie einen richtigen Schwellwert θ anhand von positiven und negativen Beispielen zu finden. Der vorgeschlagene Ansatz, RAVEN, geht davon aus, dass σ eine vorgegebene (z. B. konjunktive) Kombination einer Menge von Ähnlichkeitsfunktionen ist. Auf einer Lösung des Hospital-Residents Problems aufbauend findet RAVEN die miteinander zu vergleichenden Eigenschaften von Ressourcen. Anstatt einen Batch-Learning Ansatz zu verfolgen, kommt hier aktives Lernen zum Einsatz (Settles, 2012). Dieser Lernansatz basiert auf der Idee von neugierigen Klassifikationsverfahren, welche Fragen stellen können, um die fÃŒr die menschliche Klassifikation am besten geeigneten Paare (s, t) zu finden. Damit reduzieren aktive Lernansätze die Anzahl der zum Erzielen einer hohen Genauigkeit sowie einer hohen Vollständigkeit benötigten Beispiele. Die Auswahl der Paare (s,t) erfolgt im Falle dieses Verfahrens über die Messung der Distanz zwischen (s,t) und den Trennebenen zwischen den Klassen $+1$ und -1. Die Evaluation von RAVEN zeigt, dass weniger als 50 Fragen benötigt werden, um hohe F-Measure-Werte erreichen zu können. Diese Menge von Trainingsbeispielen ist wesentlich kleiner, als die für den damaligen Stand der Technik benötigte Menge.

Kapitel 12 nimmt sich die Schwächen des RAVEN-Verfahrens an und untersucht die Generierung von Klassifikationsfunktionen, deren Form nicht vorab bekannt ist (Ngonga Ngomo and Lyko, 2012). Das Link Discovery Problem wird hier ebenso als Klassifikationsproblem modelliert, mit dem Unterschied, dass die Klassifikationsfunktionen über genetische Programmierungsmechanismen errechnet werden. Hierbei beruht die Intuition darauf, dass komplexe Ähnlichkeitsfunktionen durch die Kombination von Genomen anderer Funktionen generativ erzeugt werden können. Der EAGLE-Ansatz beginnt mit einer kleinen Menge von positiven und negativen Beispielen. In einem zweiten Schritt wird eine zufällige Population von Ähnlichkeitsfunktionen erzeugt. Schritt 3 umfasst die Erzeugung neuer Individuen über Operationen wie Mutation und Crossover. Dabei generiert der Mutationsoperator neue Ähnlichkeitsfunktionen durch die zufällige Veränderung einer genutzten atomaren Metrik oder eines Schwellwerts in einem zufällig gewählten Individuum. Der Crossover Operator selektiert jeweils einen Unterbaum von zwei Ähnlichkeitsfunktionen und tauscht sie aus. Die Fitness der Individuen wird über ihren F-

Measure-Wert bezüglich der bekannten positiven und negativen Beispiele ermittelt. Die Individuen mit der schlechtesten Fitness werden gelöscht und die Evolution iterativ wiederholt. Da EAGLE auch aktives Lernen implementiert, bedarf es nun der Auswahl von Paaren, deren Klassifikation es zu erfragen gilt. Dazu wählt EAGLE die Paare mit der höchsten Entropie aus, d. h., die Paare (s, t), welche die Funktion $n(N - n)$ maximieren, wo n die Anzahl der Individuen, die (s, t) mit $+1$ klassifizieren und N die Größe der Population ist. Die vom Nutzer klassifizierten Paare werden zur Trainingsmenge hinzugefügt und der Ansatz geht zurück zum Evolutionsschritt. Das iterative Trainieren und Bewerten wird solange durchgeführt, bis ein Terminierungskriterium erfüllt ist (Anzahl der Iterationen, F-Measure usw.). Die Evaluation von EAGLE zeigte, dass der Ansatz dem damaligen Stand der Technik besonders hinsichtlich seiner Laufzeit aber auch bezüglich der erzielten Genauigkeit und Vollständigkeit überlegen war.

Eine Frage war, ob das Konvergenzverhalten von EAGLE über eine bessere Auswahl von Beispielen gesteigert werden kann. Eine implizite Annahme der Funktion $n(N - n)$ ist die Unabhängigkeit der Beispiele. Wird diese Unabhängigkeit angenommen, dann beeinflusst die Auswahl eines Beispiels die Auswahl eines anderen Beispiels nicht (Ngonga Ngomo et al., 2013). Kapitel 13 baut auf Kapitel 12 auf und erörtert die Berücksichtigung von Korrelationen zwischen Beispielen bei der Auswahl von Fragen für den Nutzer. Hier werden Bespiele als Vektoren im Raum aller atomaren Ähnlichkeiten betrachtet, welche zu den Individuen in der derzeitigen Population eines genetischen Lernverfahrens (hier EAGLE) gehören. Aus dieser Darstellung ergibt sich, dass jedes Beispiel eine Ähnlichkeit mit anderen Beispielen aufweist. Die Auswahl eines Beispiels bedingt dementsprechend die Auswahl von anderen Beispielen. Zwei Verfahren zur Nutzung dieser Eigenschaft von Beispielen werden vorgestellt. Das eine basiert auf Clustering und zielt darauf ab, Gruppen von ähnlichen Beispielen zu finden. Aus diesen Gruppen werden Zentroide ausgewählt und zur manuellen Bewertung verwendet. Der andere Ansatz arbeitet nach dem Prinzip der Diffusion von Ähnlichkeit durch ein Netzerk und dient ebenso zur Auswahl von Beispielen, die Zentroiden ähneln. Die Evaluation zeigt, dass die Berücksichtigung von Korrelationen dem EAGLE-Ansatz zu einer schnelleren Konvergenz verhilft. Bei der Nutzung des Ansatzes ist aber auch mit entsprechenden Skalierbarkeitseinbußen zu rechnen, da die Auswahl der Beispiele nun rechenintensiver wird.

Kapitel 14 geht einen Schritt weiter und stellt die Frage, ob Trainingsbeispiele wirklich benötigt werden (Ngonga Ngomo and Lyko, 2013). Es wird gezeigt, dass unüberwachtes maschinelles Lernen eingesetzt werden kann, wenn es sich bei der zu berechnenden Relation um eine Äquivalenzrelation handelt. In diesem Falle ist es ausreichend, eine Pseudo-F-Measure zu maximieren. Ein auf hierarchischer Suche basierender sowie deterministischer Ansatz namens EUCLID wird erarbeitet und mit einer unüberwachten Version von EAGLE verglichen. Das Kapitel zeigt, dass die Kombination von EUCLID und EAGLE mit Pseudo-F-Measures zu Ergebnissen führt, welche mit denen von überwacht trainierten Verfahren vergleichbar sind.

Kapitel 15 erweitert das Link Discovery Problem um die konkurrente Analyse von mehr als zwei Wissensbasen. Die Idee hier dem COLIBRI-Ansatz ist die Nutzung von Link Discovery in Kombination mit automatisierten Reparaturmechanismen für RDF-Wissensbasen mit dem Ziel, mehrere Wissensbasen gleichzeitig miteinander zu verlinken. Ausgehend von einer Pseudo-F-Measure werden zunächst

Links zwischen Paaren von Wissensbasen errechnet. Dazu wird die unüberwachte Variante von EAGLE verwendet. Es wird vorausgesetzt, dass die zu lernende Relation zwischen den Wissensbasen transitiv ist. Diese Eigenschaft wird genutzt, um nach möglichen Inkonsistenzen in den Wissensbasen zu suchen. Nach der automatischen Behebung der eindeutigen Inkonsistenzen wird die Linking-Prozedur wiederholt, bis keine Inkonsistenzen mehr gefunden werden können. Der COLIBRI Ansatz ist der erste unüberwachte Link Discovery Ansatz, welcher Wissensbasen miteinander verknüpfen und während des Linkingprozesses auch reparieren kann.

Die Evaluation der Performanz weiterer maschineller Lernverfahren ist der Kern von Kapitel 16. Oft genutzte Implementierungen von zehn maschinellen Lernverfahren werden mithilfe von Benchmarkdatensätzen verglichen. Dabei werden sowohl die Skalierbarkeit als auch die Genauigkeit und die Vollständigkeit der Algorithmen verglichen. Abgerundet wird der zweite Teil der Arbeit durch Kapitel 17, worin ein auf Refinement Operatoren basierender Ansatz zum Lernen von Key-Properties (dt.: Schüsseleigenschaften, Übersetzung des Verfassers) erarbeitet wird. Die theoretischen Eigenschaften des Operators werden zur Erarbeitung einer effizienten Implementierung des Ansatzes genutzt. Ein Vergleich mit anderen Verfahren zeigt, dass das bessere theoretische Rahmenwerk hinter dem Operator durch eine skalierbarere Implementierung vervollständigt wird, was zu vollständigeren Schlüsseln führt.

30.5 DAS LIMES-FRAMEWORK

Teil IV dieser Habilitationsschrift stellt das LIMES-Framework vor, worin die in Teil II und III vorgestellten Ansätze implementiert wurden. Kapitel 18 geht auf die Nutzung des Werkzeugs ein. Vorgestellt werden die Möglichkeiten zu Konfigurieren des Werkzeugs. Es wird besonders auf die Komponenten einer Link-Spezifikation eingegangen, welche zur Konfiguration des Frameworks genutzt werden kann. Von besonderer Bedeutung ist die Spezifikation der zu verwendenden Ähnlichkeitsfunktion. Es werden sowohl atomare Funktionen als auch Operatoren zum Verknüpfen von atomaren Funktionen vorgestellt. Die Semantik der Operatoren (die mengentheoretischer Natur sind) wird definiert. Es wird durchgehend beispiel basiert gezeigt, wie das Framework zu verwenden ist.

An LIMES gekoppelt sind mehrere grafische Benutzerschnittstellen (engl.: GUI, graphical user interface). Kapitel 19 stellt die Online-Schnittstelle SAIM vor. Es wird auf die Architektur hinter der Schnittstelle sowie auf die Führung des Nutzers durch den Prozess der Erstellung von Link Spezifikationen eingegangen. Zusätzlich zur expliziten Erstellung von Spezifikationen unterstützt die Oberfläche auch die Nutzung der maschinellen Lernverfahren in LIMES.

Zur Speicherung und Verwaltung der Ergebnisse von LIMES sowie von anderen Link Discovery Frameworks wurde das LINKLION repository (dt.: Ablagesystem, Übersetzung des Verfassers) konzipiert und implementiert. Kapitel 20 gibt einen Überblick über die verwendete Ontologie sowie über Anwendungfälle für das System. Die LINKLION-Ontologie verwendet eine Vielzahl existierender Ontologien wieder und unterstützt damit sowohl die Ablage von Links als auch von Metadaten zu Links. Durch die Repräsentation der Gesamtheit der Daten mittels RDF können die gespeicherten Links über SPARQL abgefragt werden. Damit werden eine Vielzahl neuer Anwendungen rund um Links möglich, eine Auswahl derer im Kapitel aufgezeigt wird.

30.6 ANWENDUNGEN

Der letzte inhaltliche Teil dieser Habilitationsschrift stellt Anwendungen der hier erarbeiteten Algorithmen in verschiedenen Domänen vor. Die erste Anwendung ist der mit dem Best Research Paper Award der International Semantic Web Conference 2011 ausgezeichnete Ansatz zur Generierung von SPARQL-Benchmarks aus Anfragelogs. Die Mehrzahl der Benchmarks zur Evaluation von Speicherungslösungen für RDF-Daten basieren auf synthetischen Daten sowie auf synthetischen Anfragen. Der wesentliche Vorteil solcher Benchmarks ist, dass eine große Datenmenge zur Laufzeit des Benchmarks erzeugt und ausgewertet werden kann. Damit können Systeme auf ihre Skalierbarkeit hin untersucht werden. Die von synthetischen Benchmarks erzeugten Daten sind jedoch zum größten Teil anders strukturiert als native RDF-Daten. Damit unterscheidet sich die Performanz von Triple Stores in echten Anwendungen signifikant von der mit diesen Benchmarks vorhergesagten Performanz. Ziel von Kapitel 21 ist die Erarbeitung eines neuen Verfahrens zur Generierung von Benchmarks für Triple Stores aus echten Daten und echten Anfragen. Der LIMES Algorithmus aus Kapitel 3 spielt dabei eine Schlüsselrolle, da er das zeiteffiziente Clustering der echten Anfragen überhaupt möglich macht. Die Ergebnisse des Kapitels zeigen neue Seiten von Triple Stores auf.

Kapitel 22 stellt den FEASIBLE-Ansatz zur Generierung von Benchmarks vor. Im Gegensatz zum vorherigen Ansatz ermöglicht FEASIBLE die Generierung von auf Anwendungen zugeschnittenen Benchmarks. Zu diesem Zweck werden 16 wichtige Eigenschaften aus SPARQL Anfragen errechnet und zum Clustering dieser Anfragen verwendet. Der exemplarbasierte Ansatz von LIMES wird auch hier eingesetzt und zur Exploration des Raumes der zu clusternden Anfragen verwendet. Das resultierende Benchmark zeigt auf, dass frühere Annahmen zur Skalierbarkeit von Triple Stores nur bedingt zutreffen.

Die DEQA-Anwendung zeigt eine weitere Stärke des LIMES-Frameworks auf. Wie in Kapitel 23 detailliert beschrieben, werden für komplexe domänenspezifische Fragenbeantwortungssysteme verlinkte Daten benötigt. Die Arten der zu generierenden Links variieren in Abhängigkeit der Domäne. Im Falle von DEQA wurden geo-spatiale Links zur Verlinkung von Instanzen von Schulen und von Mietwohungen benötigt. Der in Kapitel 6 erarbeitete skalierbare Ansatz wurde zur Erzeugung der benötigten Daten eingesetzt, wodurch die Beantwortung von komplexen Fragen zu Eigenschaften von Mietwohungen sowie zu ihrer Nähe zu anderen geographischen Entitäten möglich wurden.

Vom LIMES-Framework angebotenen Implementierungen von Metriken zum Vergleich von Zeichenketten werden in Kapitel 24 angewendet. Hier war das Ziel, eine offene Relationsextraktionsplattform zu schaffen. Dafür notwendig war es, Sätze aus großen Textkorpora zu extrahieren und anschließend auf die Existenz von Entitäten hin zu untersuchen. Die Ähnlichkeit von signifikant häufig auftretenden Satzmustern zwischen Entitäten war zum anschließenden Clustering der Muster zu berechnen. Naive Ansätze hätten dazu geführt, dass die Plattform nicht über die Skalierbarkeit verfügt, welche zur Verarbeitung von großen Datenströmen benötigt wird. Durch den Einsatz von LIMES konnte das Clustering in der Plattform und somit die auch die gesamte Plattform dazu gebracht werden, tausende RSS-Feeds pro Stunde zu analysieren.

In den Kapiteln 25, 26 und 27 dient LIMES der Verknüpfung von neuen Datensätzen mit bereits in der Linked Open Data Cloud existierenden Wissensbasen.

Linked TCGA (The Cancer Genome Atlas) war zur Zeit seiner Erzeugung der größte Datensatz in der Linked Data Cloud mit mehr als 20 Milliarden Fakten. Millionen dieser Fakten wurden von LIMES erzeugt, unter anderen im Rahmen der Verknüpfung des neuen Datensatzes mit Wissensbasen aus Bio2RDF.[21] Linked SDMX (Linked Statistic Data and Metadata Exchange) ist ein weiterer Datensatz, der über LIMES mit der Linked Open Data Cloud verknüpft wurde. Hier wurden vorrangig `owl:sameAs` Links berechnet. Der Semantic Quran Datensatz nutzte mehr von der LIMES-Funktionalität und erzeugte neben `owl:sameAs` Links auch linguistische Links zu RDF-Ressourcen aus der Linguistic Linked Open Data Cloud.

Die letzten Kapitel der Arbeit, Kapitel 28, 29 sowie 30 fassen die Arbeit zusammen und gehen auf zukünftige Arbeiten ein.

21 http://bio2rdf.org/

BIBLIOGRAPHY

Acosta, M., M. Vidal, T. Lampo, J. Castillo, and E. Ruckhaus (2011). ANAPSID: an adaptive query processing engine for SPARQL endpoints. In *The Semantic Web - ISWC 2011 - 10th International Semantic Web Conference, Bonn, Germany, October 23-27, 2011, Proceedings, Part I*, pp. 18–34.

Agrawal, R., T. Imielinski, and A. N. Swami (1993). Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, Washington, D.C., May 26-28, 1993.*, pp. 207–216.

Aluç, G., O. Hartig, M. T. Özsu, and K. Daudjee (2014). Diversified stress testing of RDF data management systems. In *The Semantic Web - ISWC 2014 - 13th International Semantic Web Conference, Riva del Garda, Italy, October 19-23, 2014. Proceedings, Part I*, pp. 197–212.

Arasu, A., M. Götz, and R. Kaushik (2010). On active learning of record matching packages. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2010, Indianapolis, Indiana, USA, June 6-10, 2010*, pp. 783–794.

Arasu, A., C. Ré, and D. Suciu (2009). Large-scale deduplication with constraints using dedupalog. In *Proceedings of the 25th International Conference on Data Engineering, ICDE 2009, March 29 2009 - April 2 2009, Shanghai, China*, pp. 952–963.

Araujo, S., J. Hidders, D. Schwabe, de Vries, and A. P. (2011, July 06). SERIMI - resource description similarity, RDF instance matching and interlinking.

Arias, M., J. D. Fernández, M. A. Martínez-Prieto, and P. de la Fuente (2011). An empirical study of real-world SPARQL queries. *CoRR abs/1103.5043*.

Asanovic, K., R. Bodik, B. C. Catanzaro, J. J. Gebis, P. Husbands, K. Keutzer, D. A. Patterson, W. L. Plishker, J. Shalf, S. W. Williams, and K. A. Yelick (2006, Dec). The Landscape of Parallel Computing Research: A View from Berkeley. Technical Report UCB/EECS-2006-183, EECS Department, University of California, Berkeley.

Atallah, M. J. (1983). A linear time algorithm for the hausdorff distance between convex polygons. Technical report, Purdue University, Department of Computer Science.

Atallah, M. J., C. C. Ribeiro, and S. Lifschitz (1991). Computing some distance functions between polygons. *Pattern Recognition 24*(8), 775–781.

Atencia, M., J. David, and J. Euzenat (2014). Data interlinking through robust linkkey extraction. In T. Schaub, G. Friedrich, and B. O'Sullivan (Eds.), *Proc. 21st european conference on artificial intelligence (ECAI), Praha (CZ)*, Amsterdam (NL), pp. 15–20. IOS press.

Auer, S., C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. G. Ives (2007). Dbpedia: A nucleus for a web of open data. In *The Semantic Web, 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference, ISWC 2007 + ASWC 2007, Busan, Korea, November 11-15, 2007.*, pp. 722–735.

Auer, S., J. Demter, M. Martin, and J. Lehmann (2012). Lodstats - an extensible framework for high-performance dataset analytics. In *Knowledge Engineering and Knowledge Management - 18th International Conference, EKAW 2012, Galway City, Ireland, October 8-12, 2012. Proceedings*, pp. 353–362.

Auer, S., J. Lehmann, and S. Hellmann (2009). Linkedgeodata: Adding a spatial dimension to the web of data. In *The Semantic Web - ISWC 2009, 8th International Semantic Web Conference, ISWC 2009, Chantilly, VA, USA, October 25-29, 2009. Proceedings*, pp. 731–746.

Auer, S., J. Lehmann, A.-C. Ngonga Ngomo, and A. Zaveri (2013a). Introduction to linked data and its lifecycle on the web. In *Reasoning Web. Semantic Technologies for Intelligent Data Access - 9th International Summer School 2013, Mannheim, Germany, July 30 - August 2, 2013. Proceedings*, pp. 1–90. Springer.

Auer, S., J. Lehmann, A.-C. Ngonga Ngomo, and A. Zaveri (2013b). Introduction to linked data and its lifecycle on the web. In *Reasoning Web. Semantic Technologies for Intelligent Data Access - 9th International Summer School 2013, Mannheim, Germany, July 30 - August 2, 2013. Proceedings*, pp. 1–90.

Augenstein, I., S. Padó, and S. Rudolph (2012). Lodifier: Generating linked data from unstructured text. In *The Semantic Web: Research and Applications - 9th Extended Semantic Web Conference, ESWC 2012, Heraklion, Crete, Greece, May 27-31, 2012. Proceedings*, pp. 210–224.

Balcan, M., A. Blum, and N. Srebro (2008). Improved guarantees for learning via similarity functions. In *21st Annual Conference on Learning Theory - COLT 2008, Helsinki, Finland, July 9-12, 2008*, pp. 287–298.

Bartoň, M., I. Hanniel, G. Elber, and M.-S. Kim (2010, November). Precise hausdorff distance computation between polygonal meshes. *Comput. Aided Geom. Des. 27*(8), 580–591.

Bayardo, R. J., Y. Ma, and R. Srikant (2007). Scaling up all pairs similarity search. In *Proceedings of the 16th International Conference on World Wide Web, WWW 2007, Banff, Alberta, Canada, May 8-12, 2007*, pp. 131–140.

Bell, G., T. Hey, and A. Szalay (2009). Beyond the data deluge. pp. 1297–1298.

Bellahsene, Z., A. Bonifati, and E. Rahm (Eds.) (2011). *Schema Matching and Mapping*. Data-Centric Systems and Applications. Springer.

Belleau, F., M.-A. Nolin, N. Tourigny, P. Rigault, and J. Morissette (2008). Bio2rdf: Towards a mashup to build bioinformatics knowledge systems. *Journal of Biomedical Informatics 41*(5), 706–716.

Bellet, A., A. Habrard, and M. Sebban (2011). Learning good edit similarities with generalization guarantees. In *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2011, Athens, Greece, September 5-9, 2011. Proceedings, Part I*, pp. 188–203.

Bellet, A., A. Habrard, and M. Sebban (2012). Good edit similarity learning by loss minimization. *Machine Learning 89*(1-2), 5–35.

Ben-David, D., T. Domany, and A. Tarem (2010). Enterprise data classification using semantic web technologies. In *The Semantic Web - ISWC 2010 - 9th International Semantic Web Conference, ISWC 2010, Shanghai, China, November 7-11, 2010, Revised Selected Papers, Part II*, pp. 66–81.

Bennett, K. P., M. C. Ferris, and Y. E. Ioannidis (1991). A genetic algorithm for database query optimization. In *Proceedings of the 4th International Conference on Genetic Algorithms, San Diego, CA, USA, July 1991*, pp. 400–407.

Berners-Lee, T., J. Hendler, O. Lassila, et al. (2001). The semantic web. *Scientific american 284*(5), 28–37.

Bernt, M., A. Donath, F. Jühling, F. Externbrink, C. Florentz, G. Fritzsch, J. Pütz, M. Middendorf, and P. F. Stadler (2012). `MITOS`: Improved *de novo* metazoan mitochondrial genome annotation. *Mol. Phylog. Evol. 69*, 313–319.

Bhagdev, R., S. Chapman, F. Ciravegna, V. Lanfranchi, and D. Petrelli (2008). Hybrid search: Effectively combining keywords and semantic searches. In *The Semantic Web: Research and Applications, 5th European Semantic Web Conference, ESWC 2008, Tenerife, Canary Islands, Spain, June 1-5, 2008, Proceedings*, pp. 554–568.

Bilenko, M. and R. J. Mooney (2003). Adaptive duplicate detection using learnable string similarity measures. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, August 24 - 27, 2003*, pp. 39–48.

Bishop, B., A. Kiryakov, D. Ognyanoff, I. Peikov, Z. Tashev, and R. Velkov (2011). OWLIM: A family of scalable semantic repositories. *Semantic Web 2*(1), 33–42.

Bizer, C., T. Heath, and T. Berners-Lee (2009). Linked data - the story so far. *Int. J. Semantic Web Inf. Syst. 5*(3), 1–22.

Bizer, C., J. Lehmann, G. Kobilarov, S. Auer, C. Becker, R. Cyganiak, and S. Hellmann (2009). Dbpedia - A crystallization point for the web of data. *J. Web Sem. 7*(3), 154–165.

Bizer, C. and A. Schultz (2009). The Berlin SPARQL Benchmark. *Int. J. Semantic Web Inf. Syst. 5*(2), 1–24.

Bizer, C. and A. Schultz (2010). The R2R Framework: Publishing and Discovering Mappings on the Web. In *COLD Workshop*.

Bleiholder, J. and F. Naumann (2008). Data fusion. *ACM Comput. Surv. 41*(1), 1:1–1:41.

Bloehdorn, S. and Y. Sure (2007). Kernel methods for mining instance data in ontologies. In *The Semantic Web, 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference, ISWC 2007 + ASWC 2007, Busan, Korea, November 11-15, 2007.*, pp. 58–71.

Bodó, Z., Z. Minier, and L. Csató (2011). Active learning with clustering. In *Active Learning and Experimental Design workshop, In conjunction with AISTATS 2010, Sardinia, Italy, May 16, 2010*, pp. 127–139.

Böhm, C., G. de Melo, F. Naumann, and G. Weikum (2012). LINDA: distributed web-of-data-scale entity matching. In *21st ACM International Conference on Information and Knowledge Management, CIKM'12, Maui, HI, USA, October 29 - November 02, 2012*, pp. 2104–2108.

Botelho, F. C. and N. Ziviani (2007). External perfect hashing for very large key sets. In *Proceedings of the Sixteenth ACM Conference on Information and Knowledge Management, CIKM 2007, Lisbon, Portugal, November 6-10, 2007*, pp. 653–662.

Broekstra, J., A. Kampman, and F. van Harmelen (2002). Sesame: A generic architecture for storing and querying RDF and RDF schema. In *The Semantic Web - ISWC 2002, First International Semantic Web Conference, Sardinia, Italy, June 9-12, 2002, Proceedings*, pp. 54–68.

Brohée, S. and J. van Helden (2006). Evaluation of clustering algorithms for protein-protein interaction networks. *BMC Bioinformatics 7*, 488.

Bühmann, L. and J. Lehmann (2012). Universal OWL axiom enrichment for large knowledge bases. In *Knowledge Engineering and Knowledge Management - 18th International Conference, EKAW 2012, Galway City, Ireland, October 8-12, 2012. Proceedings*, pp. 57–71.

Capadisli, S. (2012). Statistical linked dataspaces. Master's thesis, National University of Ireland.

Capadisli, S., S. Auer, and A.-C. Ngonga Ngomo (2015). Linked SDMX data: Path to high fidelity statistical linked data. *Semantic Web 6*(2), 105–112.

Chang, C., M. Kayed, M. R. Girgis, and K. F. Shaalan (2006). A survey of web information extraction systems. *IEEE Trans. Knowl. Data Eng. 18*(10), 1411–1428.

Chaudhuri, S. (1998). An overview of query optimization in relational systems. In *Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, PODS '98, pp. 34–43. ACM.

Cheatham, M. and P. Hitzler (2013). String similarity metrics for ontology alignment. In *The Semantic Web - ISWC 2013 - 12th International Semantic Web Conference, Sydney, NSW, Australia, October 21-25, 2013, Proceedings, Part II*, pp. 294–309.

Chen, L. T. and D. Bairagi (2010, September). Developing Parallel Programs – A Discussion of Popular Models. Technical report, Oracle Corporation.

Chiang, R. H. L., T. M. Barron, and V. C. Storey (1994). Reverse engineering of relational databases: Extraction of an EER model from a relational database. *Data Knowl. Eng. 12*(2), 107–142.

Chin, L., W. C. Hahn, G. Getz, and M. Meyerson (2011). Making sense of cancer genomic data. *Genes & development 25*(6), 534–555.

Christen, P. (2008). Febrl: a freely available record linkage system with a graphical user interface. In *Proceedings of the second Australasian workshop on Health data and knowledge management - Volume 80*, HDKM '08, Darlinghurst, Australia, Australia, pp. 17–25. Australian Computer Society, Inc.

Cilibrasi, R. and P. M. B. Vitányi (2005). Clustering by compression. *IEEE Transactions on Information Theory 51*(4), 1523–1545.

Cristianini, N. and E. Ricci (2008). Support vector machines. In *Encyclopedia of Algorithms*.

Cruz, I. F., C. Stroe, M. Caci, F. Caimi, M. Palmonari, F. P. Antonelli, and U. C. Keles (2010). Using agreementmaker to align ontologies for OAEI 2010. In *Proceedings of the 5th International Workshop on Ontology Matching (OM-2010), Shanghai, China, November 7, 2010*.

Cudré-Mauroux, P., P. Haghani, M. Jost, K. Aberer, and H. de Meer (2009). idmesh: graph-based disambiguation of linked data. In *Proceedings of the 18th International Conference on World Wide Web, WWW 2009, Madrid, Spain, April 20-24, 2009*, pp. 591–600.

d'Amato, C., N. Fanizzi, and F. Esposito (2008). Non-parametric statistical learning methods for inductive classifiers in semantic knowledge bases. In *Proceedings of the 2th IEEE International Conference on Semantic Computing (ICSC 2008), August 4-7, 2008, Santa Clara, California, USA*, pp. 291–298.

Davidov, D. and A. Rappoport (2008). Classification of semantic relationships between nominals using pattern clusters. In *ACL 2008, Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics, June 15-20, 2008, Columbus, Ohio, USA*, pp. 227–235.

de Carvalho, M. G., A. H. F. Laender, M. A. Gonçalves, and A. S. da Silva (2008). Replica identification using genetic programming. In *Proceedings of the 2008 ACM Symposium on Applied Computing (SAC), Fortaleza, Ceara, Brazil, March 16-20, 2008*, pp. 1801–1806.

de Freitas, J., G. L. Pappa, A. S. da Silva, M. A. Gonçalves, E. S. de Moura, A. Veloso, A. H. F. Laender, and M. G. de Carvalho (2010). Active learning genetic programming for record deduplication. In *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2010, Barcelona, Spain, 18-23 July 2010*, pp. 1–8.

Dean, J. and S. Ghemawat (2004). MapReduce: simplified data processing on large clusters. In *Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, Berkeley, CA, USA, pp. 10–10. USENIX Association.

Dean, J. and S. Ghemawat (2008). Mapreduce: simplified data processing on large clusters. *Commun. ACM 51*(1), 107–113.

Deus, H. F., D. F. Veiga, P. R. Freire, J. N. Weinstein, G. B. Mills, and J. S. Almeida (2010). Exposing the cancer genome atlas as a SPARQL endpoint. *Journal of Biomedical Informatics 43*(6), 998–1008.

Doan, A., J. Madhavan, R. Dhamankar, P. M. Domingos, and A. Y. Halevy (2003). Learning to match ontologies on the semantic web. *VLDB J. 12*(4), 303–319.

Dreßler, K. and A.-C. Ngonga Ngomo (2014). Time-efficient execution of bounded jaro-winkler distances. In *Proceedings of the 9th International Workshop on Ontology Matching collocated with the 13th International Semantic Web Conference (ISWC 2014), Riva del Garda, Trentino, Italy, October 20, 2014.*, pp. 37–48.

Droste, M. and P. Gastin (2009). Weighted automata and weighted logics. In *Handbook of weighted automata*, pp. 175–211. Springer.

Duan, S., A. Kementsietsidis, K. Srinivas, and O. Udrea (2011). Apples and oranges: a comparison of RDF benchmarks and real RDF datasets. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2011, Athens, Greece, June 12-16, 2011*, pp. 145–156.

Duggan, M. (2013). Cell phone activities 2013.

Ebnenasir, A. and R. Beik (2009). Developing parallel programs: A design-oriented perspective. In *Proceedings of the 2009 ICSE Workshop on Multicore Software Engineering*, IWMSE '09, Washington, DC, USA, pp. 1–8. IEEE Computer Society.

Eisenberg, V. and Y. Kanza (2012). D2rq/update: updating relational data via virtual RDF. In *Proceedings of the 21st World Wide Web Conference, WWW 2012, Lyon, France, April 16-20, 2012 (Companion Volume)*, pp. 497–498.

El-Ghazawi, T. and F. Cantonnet (2002). Upc performance and potential: a npb experimental study. In *Supercomputing*, Los Alamitos, CA, USA, pp. 1–26. IEEE Computer Society Press.

Elfeky, M. G., A. K. Elmagarmid, and V. S. Verykios (2002). TAILOR: A record linkage tool box. In *Proceedings of the 18th International Conference on Data Engineering, San Jose, CA, USA, February 26 - March 1, 2002*, pp. 17–28.

Elmagarmid, A. K., P. G. Ipeirotis, and V. S. Verykios (2007). Duplicate record detection: A survey. *IEEE Trans. Knowl. Data Eng. 19*(1), 1–16.

Erling, O. and I. Mikhailov (2007). RDF support in the virtuoso DBMS. In *The Social Semantic Web 2007, Proceedings of the 1st Conference on Social Semantic Web (CSSW), September 26-28, 2007, Leipzig, Germany.*, pp. 59–68.

Etzioni, O., M. Banko, S. Soderland, and D. S. Weld (2008). Open information extraction from the web. *Commun. ACM 51*(12), 68–74.

Etzioni, O., M. Cafarella, D. Downey, A.-M. Popescu, T. Shaked, S. Soderland, D. S. Weld, and A. Yates (2005, June). Unsupervised named-entity extraction from the web: an experimental study. *Artif. Intell. 165*, 91–134.

Euzenat, J. (2008). Algebras of ontology alignment relations. In *The Semantic Web - ISWC 2008, 7th International Semantic Web Conference, ISWC 2008, Karlsruhe, Germany, October 26-30, 2008. Proceedings*, pp. 387–402.

Euzenat, J., A. Ferrara, C. Meilicke, J. Pane, F. Scharffe, P. Shvaiko, H. Stuckenschmidt, O. Sváb-Zamazal, V. Svátek, and C. T. dos Santos (2010). Results of

the ontology alignment evaluation initiative 2010. In *Proceedings of the 5th International Workshop on Ontology Matching (OM-2010), Shanghai, China, November 7, 2010*.

Euzenat, J., A. Ferrara, W. R. van Hage, L. Hollink, C. Meilicke, A. Nikolov, D. Ritze, F. Scharffe, P. Shvaiko, H. Stuckenschmidt, O. Sváb-Zamazal, and C. T. dos Santos (2011). Results of the ontology alignment evaluation initiative 2011. In *Proceedings of the 6th International Workshop on Ontology Matching, Bonn, Germany, October 24, 2011*.

Fader, A., S. Soderland, and O. Etzioni (2011). Identifying relations for open information extraction. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing, EMNLP 2011, 27-31 July 2011, John McIntyre Conference Centre, Edinburgh, UK, A meeting of SIGDAT, a Special Interest Group of the ACL*, pp. 1535–1545.

Fanizzi, N., C. d'Amato, and F. Esposito (2009). Inductive classification of semantically annotated resources through reduced coulomb energy networks. *Int. J. Semantic Web Inf. Syst. 5*(4), 19–38.

Farrar, S. and T. Langendoen (2003). A linguistic ontology for the semantic web. *GLOT INTERNATIONAL 7*.

Feng, J., J. Wang, and G. Li (2012, August). Trie-join: a trie-based method for efficient string similarity joins. *The VLDB Journal 21*(4), 437–461.

Ferrara, A., S. Montanelli, J. Noessner, and H. Stuckenschmidt (2011). Benchmarking matching applications on the semantic web. In *The Semanic Web: Research and Applications - 8th Extended Semantic Web Conference, ESWC 2011, Heraklion, Crete, Greece, May 29 - June 2, 2011, Proceedings, Part II*, pp. 108–122.

Frey, B. J. and D. Dueck (2007). Clustering by passing messages between data points. *Science 315*, 972–976.

Furche, T., G. Gottlob, G. Grasso, C. Schallhart, and A. J. Sellers (2013). Oxpath: A language for scalable data extraction, automation, and crawling on the deep web. *VLDB J. 22*(1), 47–72.

Gaag, A., A. Kohn, and U. Lindemann (2009). Function-based solution retrieval and semantic search in mechanical engineering. In *IDEC '09*, pp. 147–158.

Gale, D. and L. S. Shapley (1962). College admissions and the stability of marriage. *The American Mathematical Monthly 69*(1), 9–15.

Gärtner, T., P. Flach, and S. Wrobel (2003). On graph kernels: Hardness results and efficient alternatives. In *Learning Theory and Kernel Machines*, pp. 129–143. Springer.

Georgala, K., A. Kosmopoulos, and G. Paliouras (2014). Spam filtering: an active learning approach using incremental clustering. In *4th International Conference on Web Intelligence, Mining and Semantics (WIMS 14), WIMS '14, Thessaloniki, Greece, June 2-4, 2014*, pp. 23:1–23:12.

Georgala, K., M. A. Sherif, and A.-C. Ngonga Ngomo (2016). An efficient approach for the generation of allen relations. In *ECAI 2016 - 22nd European Conference on Artificial Intelligence, 29 August-2 September 2016, The Hague, The Netherlands - Including Prestigious Applications of Artificial Intelligence (PAIS 2016)*, pp. 948–956.

Gerber, D., S. Hellmann, L. Bühmann, T. Soru, R. Usbeck, and A.-C. Ngonga Ngomo (2013). Real-time RDF extraction from unstructured data streams. In *The Semantic Web - ISWC 2013 - 12th International Semantic Web Conference, Sydney, NSW, Australia, October 21-25, 2013, Proceedings, Part I*, pp. 135–150.

Gerber, D. and A.-C. Ngonga Ngomo (2012). Extracting multilingual natural-language patterns for RDF predicates. In *Knowledge Engineering and Knowledge Management - 18th International Conference, EKAW 2012, Galway City, Ireland, October 8-12, 2012. Proceedings*, pp. 87–96.

Getoor, L. and B. Taskar (2007). *Introduction to Statistical Relational Learning (Adaptive Computation and Machine Learning)*. The MIT Press.

Glaser, H., I. C. Millard, W.-K. Sung, S. Lee, P. Kim, and B.-J. You (2009). Research on linked data and co-reference resolution. Technical report, University of Southampton.

Goldhahn, D., T. Eckart, and U. Quasthoff (2012). Building large monolingual dictionaries at the leipzig corpora collection: From 100 to 200 languages. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC-2012), Istanbul, Turkey, May 23-25, 2012*, pp. 759–765.

Görlitz, O., M. Thimm, and S. Staab (2012). SPLODGE: systematic generation of SPARQL benchmark queries for linked open data. In *The Semantic Web - ISWC 2012 - 11th International Semantic Web Conference, Boston, MA, USA, November 11-15, 2012, Proceedings, Part I*, pp. 116–132.

Granitzer, M., V. Sabol, K. W. Onn, D. Lukose, and K. Tochtermann (2010). Ontology alignment - A survey with focus on visually supported semi-automatic techniques. *Future Internet 2*(3), 238–258.

Grant, C. E., C. P. George, J. Gumbs, J. N. Wilson, and P. J. Dobbins (2010). *Morpheus*: a deep web question answering system. In *iiWAS'2010 - The 12th International Conference on Information Integration and Web-based Applications and Services, 8-10 November 2010, Paris, France*, pp. 841–844.

Gray, J. (Ed.) (1991). *The Benchmark Handbook for Database and Transaction Systems (1st Edition)*. Morgan Kaufmann.

Guéret, C., P. T. Groth, F. van Harmelen, and S. Schlobach (2010). Finding the achilles heel of the web of data: Using network analysis for link-recommendation. In *The Semantic Web - ISWC 2010 - 9th International Semantic Web Conference, ISWC 2010, Shanghai, China, November 7-11, 2010, Revised Selected Papers, Part I*, pp. 289–304.

Gufler, B., N. Augsten, A. Reiser, and A. Kemper (2012). Load balancing in mapreduce based on scalable cardinality estimates. In *IEEE 28th International Conference on Data Engineering (ICDE 2012), Washington, DC, USA (Arlington, Virginia), 1-5 April, 2012*, pp. 522–533.

Gulhane, P., A. Madaan, R. R. Mehta, J. Ramamirtham, R. Rastogi, S. Satpal, S. H. Sengamedu, A. Tengli, and C. Tiwari (2011). Web-scale information extraction with vertex. In *Proceedings of the 27th International Conference on Data Engineering, ICDE 2011, April 11-16, 2011, Hannover, Germany*, pp. 1209–1220.

Guo, Y., Z. Pan, and J. Heflin (2005). LUBM: A benchmark for OWL knowledge base systems. *J. Web Sem. 3*(2-3), 158–182.

Guthe, M., P. Borodin, and R. Klein (2005). Fast and accurate hausdorff distance calculation between meshes. *Journal of WSCG 13*(2), 41–48.

Hartung, M., A. Groß, and E. Rahm (2013). Composition methods for link discovery. In *Datenbanksysteme für Business, Technologie und Web (BTW), 15. Fachtagung des GI-Fachbereichs "Datenbanken und Informationssysteme" (DBIS), 11.-15.3.2013 in Magdeburg, Germany. Proceedings*, pp. 261–277.

Hassan, M. M., R. Speck, and A. Ngonga Ngomo (2015). Using caching for local link discovery on large data sets. In *Engineering the Web in the Big Data Era - 15th International Conference, ICWE 2015, Rotterdam, The Netherlands, June 23-26, 2015, Proceedings*, pp. 344–354.

Hassanzadeh, O. and M. P. Consens (2009). Linked movie data base. In *Proceedings of the WWW2009 Workshop on Linked Data on the Web, LDOW 2009, Madrid, Spain, April 20, 2009*.

Hassanzadeh, O., K. Q. Pu, S. H. Yeganeh, R. J. Miller, L. Popa, M. A. Hernández, and H. Ho (2013). Discovering linkage points over web data. *PVLDB 6*(6), 444–456.

Hausenblas, M., B. Villazón-Terrazas, and R. Cyganiak (2012). Data shapes and data transformations. *CoRR abs/1211.1565*.

Heath, T. and C. Bizer (2011). *Linked Data: Evolving the Web into a Global Data Space*. Synthesis Lectures on the Semantic Web. Morgan & Claypool Publishers.

Heino, N. and J. Z. Pan (2012). RDFS reasoning on massively parallel hardware. In *The Semantic Web - ISWC 2012 - 11th International Semantic Web Conference, Boston, MA, USA, November 11-15, 2012, Proceedings, Part I*, pp. 133–148.

Hellmann, S. (2010). The semantic gap of formalized meaning. In *The Semantic Web: Research and Applications, 7th Extended Semantic Web Conference, ESWC 2010, Heraklion, Crete, Greece, May 30 - June 3, 2010, Proceedings, Part II*, pp. 462–466.

Hellmann, S., J. Lehmann, and S. Auer (2012). Linked-data aware URI schemes for referencing text fragments. In *Knowledge Engineering and Knowledge Management - 18th International Conference, EKAW 2012, Galway City, Ireland, October 8-12, 2012. Proceedings*, pp. 175–184.

Hennessy, J. L. and D. A. Patterson (2007). *Computer Architecture - A Quantitative Approach (4. ed.)*. Morgan Kaufmann.

Hillner, S. and A.-C. Ngonga Ngomo (2011). Parallelizing LIMES for large-scale link discovery. In *Proceedings the 7th International Conference on Semantic Systems, I-SEMANTICS 2011, Graz, Austria, September 7-9, 2011*, pp. 9–16.

Hogan, A., A. Polleres, J. Umbrich, and A. Zimmermann (2010). Some entities are more equal than others: statistical methods to consolidate linked data. In *Workshop on New Forms of Reasoning for the Semantic Web: Scalable & Dynamic (NeFoRS2010)*.

Hsu, F.-H., E. Serpedin, T.-H. Hsiao, A. J. Bishop, E. R. Dougherty, and Y. Chen (2012). Reducing confounding and suppression effects in tcga data: an integrated analysis of chemotherapy response in ovarian cancer. *BMC Genomics 13*(Suppl 6), S13.

Hu, W., J. Chen, G. Cheng, and Y. Qu (2010). Objectcoref & falcon-AO: results for OAEI 2010. In P. Shvaiko, J. Euzenat, F. Giunchiglia, H. Stuckenschmidt, M. Mao, and I. F. Cruz (Eds.), *Proceedings of the 5th International Workshop on Ontology Matching (OM-2010), Shanghai, China, November 7, 2010*, Volume 689 of *CEUR Workshop Proceedings*. CEUR-WS.org.

Huber, J., T. Sztyler, J. Nößner, and C. Meilicke (2011). CODI: combinatorial optimization for data integration: results for OAEI 2011. In *Proceedings of the 6th International Workshop on Ontology Matching, Bonn, Germany, October 24, 2011*.

Hudson, T. J., W. Anderson, A. Aretz, A. D. Barker, C. Bell, R. R. Bernabé, M. Bhan, F. Calvo, I. Eerola, D. S. Gerhard, et al. (2010). International network of cancer genome projects. *Nature 464*(7291), 993–998.

Ilyas, I. F., V. Markl, P. J. Haas, P. Brown, and A. Aboulnaga (2004). CORDS: automatic discovery of correlations and soft functional dependencies. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, Paris, France, June 13-18, 2004*, pp. 647–658.

Isele, R. and C. Bizer (2011). Learning linkage rules using genetic programming. In *Proceedings of the 6th International Workshop on Ontology Matching, Bonn, Germany, October 24, 2011*.

Isele, R. and C. Bizer (2012). Learning expressive linkage rules using genetic programming. *PVLDB 5*(11), 1638–1649.

Isele, R., A. Jentzsch, and C. Bizer (2011). Efficient multidimensional blocking for link discovery without losing recall. In *Proceedings of the 14th International Workshop on the Web and Databases 2011, WebDB 2011, Athens, Greece, June 12, 2011*.

Isele, R., A. Jentzsch, and C. Bizer (2012). Active learning of expressive linkage rules for the web of data. In *Web Engineering - 12th International Conference, ICWE 2012, Berlin, Germany, July 23-27, 2012. Proceedings*, pp. 411–418.

Jean-Mary, Y. R., E. P. Shironoshita, and M. R. Kabuka (2010). ASMOV: results for OAEI 2010. In *Proceedings of the 5th International Workshop on Ontology Matching (OM-2010), Shanghai, China, November 7, 2010*.

Jeong, J., L. Li, Y. Liu, K. Nephew, T. Huang, and C. Shen (2010). An empirical bayes model for gene expression and methylation profiles in antiestrogen resistant breast cancer. *BMC medical genomics 3*(1), 55.

Jiang, X., Y. Huang, M. Nickel, and V. Tresp (2012). Combining information extraction, deductive reasoning and machine learning for relation prediction. In *The Semantic Web: Research and Applications - 9th Extended Semantic Web Conference, ESWC 2012, Heraklion, Crete, Greece, May 27-31, 2012. Proceedings*, pp. 164–178.

Kamdar, M., A. Iqbal, M. Saleem, H. Deus, and S. Decker (2014). Genomesnip: Fragmenting the genomic wheel to augment discovery in cancer research. In *Conference on Semantics in Healthcare and Life Sciences (CSHALS) 2014*.

Kanne, C.-C. and G. Moerkotte (2010). Histograms reloaded: the merits of bucket diversity. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, SIGMOD '10, pp. 663–674. ACM.

Karlsson, T. J. M., Ó. T. Tirado, D. R. Barea, G. Klambauer, M. C. Castillo, and O. Trelles (2012). Enabling large-scale bioinformatics data analysis with cloud computing. In *10th IEEE International Symposium on Parallel and Distributed Processing with Applications, ISPA 2012, Leganes, Madrid, Spain, July 10-13, 2012*, pp. 640–645.

Kasim, H., V. March, R. Zhang, and S. See (2008). Survey on parallel programming model. In *Proceedings of the IFIP International Conference on Network and Parallel Computing*, NPC '08, pp. 266–275. Berlin, Heidelberg: Springer-Verlag.

Kayed, M. and C. Chang (2010). Fivatech: Page-level web data extraction from template pages. *IEEE Trans. Knowl. Data Eng. 22*(2), 249–263.

Keerthi, S. S. and C.-J. Lin (2003, July). Asymptotic behaviors of support vector machines with gaussian kernel. *Neural Comput. 15*, 1667–1689.

Kejriwal, M. and D. P. Miranker (2015). Semi-supervised instance matching using boosted classifiers. In *The Semantic Web. Latest Advances and New Domains - 12th European Semantic Web Conference, ESWC 2015, Portoroz, Slovenia, May 31 - June 4, 2015. Proceedings*, pp. 388–402.

Kernighan, M. D., K. W. Church, and W. A. Gale (1990). A spelling correction program based on a noisy channel model. In *13th International Conference on Computational Linguistics, COLING 1990, University of Helsinki, Finland, August 20-25, 1990*, pp. 205–210.

Kim, H. S., J. D. Minna, and M. A. White (2013). Gwas meets tcga to illuminate mechanisms of cancer predisposition. *Cell 152*(3), 387–389.

Kirsten, T., A. Groß, M. Hartung, and E. Rahm (2011). GOMMA: a component-based infrastructure for managing and analyzing life science ontologies and their evolution. *J. Biomedical Semantics 2*, 6.

Klyne, G. and J. J. Carroll (2004, February). Resource description framework (RDF): Concepts and abstract syntax. W3C Recommendation.

Kolb, L., A. Thor, and E. Rahm (2012a). Dedoop: Efficient deduplication with hadoop. *PVLDB 5*(12), 1878–1881.

Kolb, L., A. Thor, and E. Rahm (2012b). Load balancing for mapreduce-based entity resolution. In *IEEE 28th International Conference on Data Engineering (ICDE 2012), Washington, DC, USA (Arlington, Virginia), 1-5 April, 2012*, pp. 618–629.

Kolb, L., A. Thor, and E. Rahm (2012c). Multi-pass sorted neighborhood blocking with mapreduce. *Computer Science - R&D 27*(1), 45–63.

Kolda, T. G. and B. W. Bader (2009, August). Tensor decompositions and applications. *SIAM Rev. 51*(3), 455–500.

Kontokostas, D., P. Westphal, S. Auer, S. Hellmann, J. Lehmann, R. Cornelissen, and A. Zaveri (2014). Test-driven evaluation of linked data quality. In *23rd International World Wide Web Conference, WWW '14, Seoul, Republic of Korea, April 7-11, 2014*, pp. 747–758.

Köpcke, H. and E. Rahm (2008). Training selection for tuning entity matching. In *Proceedings of the International Workshop on Quality in Databases and Management of Uncertain Data, Auckland, New Zealand, August 2008*, pp. 3–12.

Köpcke, H., A. Thor, and E. Rahm (2009). Comparative evaluation of entity resolution approaches with fever. *Proc. VLDB Endow. 2*(2), 1574–1577.

Köpcke, H., A. Thor, and E. Rahm (2010). Evaluation of entity resolution approaches on real-world match problems. *PVLDB 3*(1), 484–493.

Koza, J. R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection (Complex Adaptive Systems)*. The MIT Press.

Kranzdorf, J., A. J. Sellers, G. Grasso, C. Schallhart, and T. Furche (2012). Visual oxpath: robust wrapping by example. In *Proceedings of the 21st World Wide Web Conference, WWW 2012, Lyon, France, April 16-20, 2012 (Companion Volume)*, pp. 369–372.

Kurtz, S. (1996). Approximate string searching under weighted edit distance. In *Proc. WSP*, Volume 96, pp. 156–170. Citeseer: Carleton University Press.

Kwon, Y., M. Balazinska, B. Howe, and J. A. Rolia (2012). Skewtune: mitigating skew in mapreduce applications. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2012, Scottsdale, AZ, USA, May 20-24, 2012*, pp. 25–36.

Lehmann, J., S. Auer, L. Bühmann, and S. Tramp (2011). Class expression learning for ontology engineering. *J. Web Sem. 9*(1), 71–81.

Lehmann, J., T. Furche, G. Grasso, A.-C. Ngonga Ngomo, C. Schallhart, A. J. Sellers, C. Unger, L. Bühmann, D. Gerber, K. Höffner, D. Liu, and S. Auer (2012). deqa: Deep web extraction for question answering. In *The Semantic Web - ISWC 2012 - 11th International Semantic Web Conference, Boston, MA, USA, November 11-15, 2012, Proceedings, Part II*, pp. 131–147.

Lehmann, J., D. Gerber, M. Morsey, and A.-C. Ngonga Ngomo (2012). Defacto - deep fact validation. In *The Semantic Web - ISWC 2012 - 11th International Semantic Web Conference, Boston, MA, USA, November 11-15, 2012, Proceedings, Part I*, pp. 312–327.

Lehmann, J. and P. Hitzler (2010). Concept learning in description logics using refinement operators. *Machine Learning 78*(1-2), 203–250.

Leser, U. and F. Naumann (2007). *Informationsintegration - Architekturen und Methoden zur Integration verteilter und heterogener Datenquellen*. dpunkt.verlag.

Levenshtein, V. I. (1965). Binary codes capable of correcting deletions, insertions, and reversals. *Doklady Akademii Nauk SSSR 163 (4)*, 845–848.

Levenshtein, V. I. (1966). Binary codes capable of correcting deletions, insertions, and reversals. Technical Report 8.

Li, G., D. Deng, J. Wang, and J. Feng (2011). PASS-JOIN: A partition-based method for similarity joins. *PVLDB 5*(3), 253–264.

Li, J., J. Tang, Y. Li, and Q. Luo (2009). Rimom: A dynamic multistrategy ontology alignment framework. *IEEE Trans. Knowl. Data Eng. 21*(8), 1218–1232.

Liere, R. and P. Tadepalli (1997). Active learning with committees for text categorization. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence and Ninth Innovative Applications of Artificial Intelligence Conference, AAAI 97, IAAI 97, July 27-31, 1997, Providence, Rhode Island.*, pp. 591–596.

Lin, D. (1998). An information-theoretic definition of similarity. In *Proceedings of the Fifteenth International Conference on Machine Learning (ICML 1998), Madison, Wisconsin, USA, July 24-27, 1998*, pp. 296–304.

Lin, J. (2002). The web as a resource for question answering: Perspectives and challenges. In *Proceedings of the Third International Conference on Language Resources and Evaluation, LREC 2002, May 29-31, 2002, Las Palmas, Canary Islands, Spain*.

Lopez, V., V. S. Uren, M. Sabou, and E. Motta (2009). Cross ontology query answering on the semantic web: an initial evaluation. In *Proceedings of the 5th International Conference on Knowledge Capture (K-CAP 2009), September 1-4, 2009, Redondo Beach, California, USA*, pp. 17–24.

Lopez, V., V. S. Uren, M. Sabou, and E. Motta (2011). Is question answering fit for the semantic web?: A survey. *Semantic Web 2*(2), 125–155.

Lyko, K., K. Höffner, R. Speck, A.-C. Ngonga Ngomo, and J. Lehmann (2013). SAIM - one step closer to zero-configuration link discovery. In *The Semantic Web: ESWC 2013 Satellite Events - ESWC 2013 Satellite Events, Montpellier, France, May 26-30, 2013, Revised Selected Papers*, pp. 167–172.

Ma, L., X. Sun, F. Cao, C. Wang, X. Wang, N. Kanellos, D. Wolfson, and Y. Pan (2009). Semantic enhancement for enterprise data management. In *The Semantic Web - ISWC 2009, 8th International Semantic Web Conference, ISWC 2009, Chantilly, VA, USA, October 25-29, 2009. Proceedings*, pp. 876–892.

Madhavan, J. and A. Y. Halevy (2003). Composing mappings among data sources. In *VLDB*, pp. 572–583.

Manlove, D., R. Irving, K. Iwama, S. Miyazaki, and Y. Morita (2002). Hard variants of stable marriage. *Theoretical Computer Science 276*(1-2), 261 – 279.

Mannila, H. and K. Räihä (1994). Algorithms for inferring functional dependencies from relations. *Data Knowl. Eng. 12*(1), 83–99.

Mannila, H. and H. Toivonen (1997). Levelwise search and borders of theories in knowledge discovery. *Data Min. Knowl. Discov. 1*(3), 241–258.

Marx, E., T. Soru, S. Shekarpour, S. Auer, A.-C. Ngonga Ngomo, and K. Breitman (2013). Towards an efficient RDF dataset slicing. *International Journal of Semantic Computing 07*(04), 455–477.

McCusker, J. P. and D. L. McGuinness (2010). Towards identity in linked data. In *Proceedings of the 7th International Workshop on OWL: Experiences and Directions (OWLED 2010), San Francisco, California, USA, June 21-22, 2010*.

Mendes, P. N., M. Jakob, A. García-Silva, and C. Bizer (2011). Dbpedia spotlight: shedding light on the web of documents. In *Proceedings the 7th International Conference on Semantic Systems, I-SEMANTICS 2011, Graz, Austria, September 7-9, 2011*, pp. 1–8.

Michelson, M. and C. A. Knoblock (2006). Learning blocking schemes for record linkage. In *Proceedings, The Twenty-First National Conference on Artificial Intelligence and the Eighteenth Innovative Applications of Artificial Intelligence Conference, July 16-20, 2006, Boston, Massachusetts, USA*, pp. 440–445.

Minack, E., W. Siberski, and W. Nejdl (2009). Benchmarking fulltext search performance of RDF stores. In *The Semantic Web: Research and Applications, 6th European Semantic Web Conference, ESWC 2009, Heraklion, Crete, Greece, May 31-June 4, 2009, Proceedings*, pp. 81–95.

Mollá, D. and J. L. V. González (2007). Question answering in restricted domains: An overview. *Computational Linguistics 33*(1), 41–61.

Morsey, M., J. Lehmann, S. Auer, and A.-C. Ngonga Ngomo (2011). Dbpedia SPARQL benchmark - performance assessment with real queries on real data. In *The Semantic Web - ISWC 2011 - 10th International Semantic Web Conference, Bonn, Germany, October 23-27, 2011, Proceedings, Part I*, pp. 454–469.

Morsey, M., J. Lehmann, S. Auer, and A.-C. Ngonga Ngomo (2012). Usage-centric benchmarking of RDF triple stores. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, July 22-26, 2012, Toronto, Ontario, Canada.*

Nakashole, N. and G. Weikum (2012). Real-time population of knowledge bases: opportunities and challenges. In *Proceedings of the Joint Workshop on Automatic Knowledge Base Construction and Web-scale Knowledge Extraction*, AKBC-WEKEX '12, Stroudsburg, PA, USA, pp. 41–45. Association for Computational Linguistics.

Nentwig, M., T. Soru, A.-C. Ngonga Ngomo, and E. Rahm (2014). Linklion: A link repository for the web of data. In *The Semantic Web: ESWC 2014 Satellite Events, Anissaras, Crete, Greece, May 25-29, 2014, Revised Selected Papers*, pp. 439–443.

Ngonga Ngomo, A. and M. M. Hassan (2016). The lazy traveling salesman–memory management for large-scale link discovery. In *The Semantic Web. Latest Advances and New Domains - 13th International Conference, ESWC 2016, Heraklion, Crete, Greece, May 29 - June 2, 2016, Proceedings*, pp. 423–438.

Ngonga Ngomo, A.-C. (2010). Parameter-free clustering of protein-protein interaction graphs. In *Proceedings of Symposium on Machine Learning in Systems Biology 2010*.

Ngonga Ngomo, A.-C. (2011). A time-efficient hybrid approach to link discovery. In *Proceedings of the 6th International Workshop on Ontology Matching, Bonn, Germany, October 24, 2011*.

Ngonga Ngomo, A.-C. (2012a). Link discovery with guaranteed reduction ratio in affine spaces with minkowski measures. In *The Semantic Web - ISWC 2012 - 11th International Semantic Web Conference, Boston, MA, USA, November 11-15, 2012, Proceedings, Part I*, pp. 378–393.

Ngonga Ngomo, A.-C. (2012b). On link discovery using a hybrid approach. *J. Data Semantics 1*(4), 203–217.

Ngonga Ngomo, A.-C. (2013). ORCHID - reduction-ratio-optimal computation of geo-spatial distances for link discovery. In *The Semantic Web - ISWC 2013 - 12th International Semantic Web Conference, Sydney, NSW, Australia, October 21-25, 2013, Proceedings, Part I*, pp. 395–410.

Ngonga Ngomo, A.-C. (2014). HELIOS - execution optimization for link discovery. In *The Semantic Web - ISWC 2014 - 13th International Semantic Web Conference, Riva del Garda, Italy, October 19-23, 2014. Proceedings, Part I*, pp. 17–32.

Ngonga Ngomo, A.-C. and S. Auer (2011). LIMES - A time-efficient approach for large-scale link discovery on the web of data. In *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*, pp. 2312–2317.

Ngonga Ngomo, A.-C., S. Auer, J. Lehmann, and A. Zaveri (2014). Introduction to linked data and its lifecycle on the web. In *Reasoning Web. Reasoning on the Web in the Big Data Era - 10th International Summer School 2014, Athens, Greece, September 8-13, 2014. Proceedings*, pp. 1–99.

Ngonga Ngomo, A.-C., N. Heino, K. Lyko, R. Speck, and M. Kaltenböck (2011). SCMS - semantifying content management systems. In *The Semantic Web - ISWC 2011 - 10th International Semantic Web Conference, Bonn, Germany, October 23-27, 2011, Proceedings, Part II*, pp. 189–204.

Ngonga Ngomo, A.-C., L. Kolb, N. Heino, M. Hartung, S. Auer, and E. Rahm (2013). When to reach for the cloud: Using parallel hardware for link discovery. In *The Semantic Web: Semantics and Big Data, 10th International Conference, ESWC 2013, Montpellier, France, May 26-30, 2013. Proceedings*, pp. 275–289.

Ngonga Ngomo, A.-C., J. Lehmann, S. Auer, and K. Höffner (2011). RAVEN - active learning of link specifications. In *Proceedings of the 6th International Workshop on Ontology Matching, Bonn, Germany, October 24, 2011*.

Ngonga Ngomo, A.-C., J. Lehmann, and M. M. Hassan (2013). Towards transfer learning of link specifications. In *2013 IEEE Seventh International Conference on Semantic Computing, Irvine, CA, USA, September 16-18, 2013*, pp. 202–205.

Ngonga Ngomo, A.-C. and K. Lyko (2012). EAGLE: efficient active learning of link specifications using genetic programming. In *The Semantic Web: Research and Applications - 9th Extended Semantic Web Conference, ESWC 2012, Heraklion, Crete, Greece, May 27-31, 2012. Proceedings*, pp. 149–163.

Ngonga Ngomo, A.-C. and K. Lyko (2013). Unsupervised learning of link specifications: deterministic vs. non-deterministic. In *Proceedings of the 8th International Workshop on Ontology Matching co-located with the 12th International Semantic Web Conference (ISWC 2013), Sydney, Australia, October 21, 2013.*, pp. 25–36.

Ngonga Ngomo, A.-C., K. Lyko, and V. Christen (2013). COALA - correlation-aware active learning of link specifications. In *The Semantic Web: Semantics and Big Data, 10th International Conference, ESWC 2013, Montpellier, France, May 26-30, 2013. Proceedings*, pp. 442–456.

Ngonga Ngomo, A.-C. and F. Schumacher (2009). Borderflow: A local graph clustering algorithm for natural language processing. In *Computational Linguistics and Intelligent Text Processing, 10th International Conference, CICLing 2009, Mexico City, Mexico, March 1-7, 2009. Proceedings*, pp. 547–558.

Ngonga Ngomo, A.-C. and F. Schumacher (2009). Borderflow: A local graph clustering algorithm for natural language processing. In *Computational Linguistics and Intelligent Text Processing, 10th International Conference, CICLing 2009, Mexico City, Mexico, March 1-7, 2009. Proceedings*, pp. 547–558.

Ngonga Ngomo, A.-C., M. A. Sherif, and K. Lyko (2014). Unsupervised link discovery through knowledge base repair. In *The Semantic Web: Trends and Challenges - 11th International Conference, ESWC 2014, Anissaras, Crete, Greece, May 25-29, 2014. Proceedings*, pp. 380–394.

Nickel, M., V. Tresp, and H. Kriegel (2012). Factorizing YAGO: scalable machine learning for linked data. In *Proceedings of the 21st World Wide Web Conference 2012, WWW 2012, Lyon, France, April 16-20, 2012*, pp. 271–280.

Nikolov, A., M. d'Aquin, and E. Motta (2012). Unsupervised learning of link discovery configuration. In *The Semantic Web: Research and Applications - 9th Extended Semantic Web Conference, ESWC 2012, Heraklion, Crete, Greece, May 27-31, 2012. Proceedings*, pp. 119–133.

Nikolov, A., V. S. Uren, E. Motta, and A. N. D. Roeck (2009). Overcoming schema heterogeneity between linked semantic repositories to improve coreference resolution. In *The Semantic Web, Fourth Asian Conference, ASWC 2009, Shanghai, China, December 6-9, 2009. Proceedings*, pp. 332–346.

Niu, X., S. Rong, Y. Zhang, and H. Wang (2011). Zhishi.links results for OAEI 2011. In *Proceedings of the 6th International Workshop on Ontology Matching, Bonn, Germany, October 24, 2011*.

Noushmehr, H., D. J. Weisenberger, K. Diefes, H. S. Phillips, K. Pujara, B. P. Berman, F. Pan, C. E. Pelloski, E. P. Sulman, K. P. Bhat, R. G. Verhaak, K. A. Hoadley, D. N. Hayes, C. M. Perou, H. K. Schmidt, L. Ding, R. K. Wilson, D. V. D. Berg, H. Shen, H. Bengtsson, P. Neuvial, L. M. Cope, J. Buckley, J. G. Herman, S. B. Baylin, P. W. Laird, and K. Aldape (2010). Identification of a cpg island methylator phenotype that defines a distinct subgroup of glioma. *Cancer Cell 17*(5), 510 – 522.

Nutanong, S., E. H. Jacox, and H. Samet (2011, May). An incremental hausdorff distance calculation algorithm. *Proc. VLDB Endow. 4*(8), 506–517.

Owens, A., N. Gibbins, and mc schraefel (2008, May). Effective benchmarking for rdf stores using synthetic data.

Owens, A., A. Seaborne, N. Gibbins, and mc schraefel (2008). Clustered TDB: A clustered triple store for jena. Technical report, Electronics and Computer Science, University of Southampton.

Pan, J. Z., J. M. Gómez-Pérez, Y. Ren, H. Wu, H. Wang, and M. Zhu (2014). Graph pattern based RDF data compression. In *Semantic Technology - 4th Joint International Conference, JIST 2014, Chiang Mai, Thailand, November 9-11, 2014. Revised Selected Papers*, pp. 239–256.

Papadakis, G., E. Ioannou, C. Niederée, T. Palpanas, and W. Nejdl (2011). Eliminating the redundancy in blocking-based entity resolution methods. In *Proceedings of the 2011 Joint International Conference on Digital Libraries, JCDL 2011, Ottawa, ON, Canada, June 13-17, 2011*, pp. 85–94.

Pedersen, T., S. Patwardhan, and J. Michelizzi (2004). Wordnet: : Similarity - measuring the relatedness of concepts. In *Proceedings of the Nineteenth National Conference on Artificial Intelligence, Sixteenth Conference on Innovative Applications of Artificial Intelligence, July 25-29, 2004, San Jose, California, USA*, pp. 1024–1025.

Pérez-Solà, C. and J. Herrera-Joancomartí (2013). Improving relational classification using link prediction techniques. In *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2013, Prague, Czech Republic, September 23-27, 2013, Proceedings, Part I*, pp. 590–605.

Pernelle, N., F. Saïs, and D. Symeonidou (2013). An automatic key discovery approach for data linking. *J. Web Sem. 23*, 16–30.

Peukert, E., H. Berthold, and E. Rahm (2010). Rewrite techniques for performance optimization of schema matching processes. In *EDBT 2010, 13th International Conference on Extending Database Technology, Lausanne, Switzerland, March 22-26, 2010, Proceedings*, pp. 453–464.

Picalausa, F. and S. Vansummeren (2011). What are real SPARQL queries like? In *Proceedings of the International Workshop on Semantic Web Information Management, SWIM 2011, Athens, Greece, June 12, 2011*, pp. 7.

Prud'hommeaux, E. and A. Seaborne (2008). SPARQL Query Language for RDF. W3C Recommendation.

Quinn, M. J. (2003). *Parallel Programming in C with MPI and OpenMP*. McGraw-Hill Education Group.

Rahm, E. and P. A. Bernstein (2001). A survey of approaches to automatic schema matching. *VLDB J. 10*(4), 334–350.

Raimond, Y., C. Sutton, and M. B. Sandler (2008). Automatic interlinking of music datasets on the semantic web. In *Proceedings of the WWW2008 Workshop on Linked Data on the Web, LDOW 2008, Beijing, China, April 22, 2008.*

Richard Cyganiak, Michael Hausenblas, E. M. (2011). *Linking Government Data*, Chapter Official statistics and the Practice of Data Fidelity. Springer.

Ristad, E. S. and P. N. Yianilos (1998). Learning string-edit distance. *IEEE Trans. Pattern Anal. Mach. Intell. 20*(5), 522–532.

Rizzo, G., R. Troncy, S. Hellmann, and M. Bruemmer (2012). NERD meets NIF: lifting NLP extraction results to the linked data cloud. In *WWW2012 Workshop on Linked Data on the Web, Lyon, France, 16 April, 2012*.

Robbins, D. E., A. Grüneberg, H. F. Deus, M. M. Tanik, and J. S. Almeida (2013). A self-updating road map of the cancer genome atlas. *Bioinformatics 29*(10), 1333–1340.

Robinson, J. T., H. Thorvaldsdóttir, W. Winckler, M. Guttman, E. S. Lander, G. Getz, and J. P. Mesirov (2011). Integrative genomics viewer. *Nature biotechnology 29*(1), 24–26.

Ruiz-Casado, M., E. Alfonseca, and P. Castells (2007). Automatising the learning of lexical patterns: An application to the enrichment of wordnet by extracting semantic relationships from wikipedia. *Data Knowl. Eng. 61*(3), 484–499.

Safavian, S. R. and D. Landgrebe (1991). A survey of decision tree classifier methodology. *Systems, Man and Cybernetics, IEEE Transactions on 21*(3), 660–674.

Saïs, F., N. Pernelle, and M. Rousset (2009). Combining a logical and a numerical method for data reconciliation. *J. Data Semantics 12*, 66–94.

Saleem, M., M. Ali, Q. Mehmood, A. Hogan, and A.-C. Ngonga Ngomo (2015). LSQ: Linked SPARQL Queries Dataset. In *The Semantic Web - ISWC 2015 - 14th International Semantic Web Conference, Bethlehem, Pennsylvania, 2015. Proceedings, Part II*.

Saleem, M., M. Kamdar, A. Iqbal, S. Sampath, and A.-C. N. Helena F. Deus (2014). Big linked cancer data: Integrating linked TCGA and pubmed. *J. Web Sem. 27*, 34–41.

Saleem, M., Q. Mehmood, and A.-C. Ngonga Ngomo (2015). FEASIBLE: A Featured-Based SPARQL Benchmarks Generation Framework. In *The Semantic Web - ISWC 2015 - 14th International Semantic Web Conference, Bethlehem, Pennsylvania, USA*.

Saleem, M. and A.-C. Ngonga Ngomo (2014). Hibiscus: Hypergraph-based source selection for SPARQL endpoint federation. In *The Semantic Web: Trends and Challenges - 11th International Conference, ESWC 2014, Anissaras, Crete, Greece, May 25-29, 2014. Proceedings*, pp. 176–191.

Saleem, M., A.-C. Ngonga Ngomo, J. X. Parreira, H. F. Deus, and M. Hauswirth (2013). DAW: duplicate-aware federated query processing over the web of data. In *The Semantic Web - ISWC 2013 - 12th International Semantic Web Conference, Sydney, NSW, Australia, October 21-25, 2013, Proceedings, Part I*, pp. 574–590.

Saleem, M., S. S. Padmanabhuni, A.-C. Ngonga Ngomo, J. S. Almeida, S. Decker, and H. F. Deus (2013). Linked cancer genome atlas database. In *I-SEMANTICS 2013 - 9th International Conference on Semantic Systems, ISEM '13, Graz, Austria, September 4-6, 2013*, pp. 129–134.

Sarawagi, S. (2008, March). Information extraction. *Found. Trends databases 1*(3), 261–377.

Sarawagi, S. and A. Bhamidipaty (2002). Interactive deduplication using active learning. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, July 23-26, 2002, Edmonton, Alberta, Canada*, pp. 269–278.

Sarawagi, S., A. Bhamidipaty, A. Kirpal, and C. Mouli (2002). ALIAS: an active learning led interactive deduplication system. In *VLDB 2002, Proceedings of 28th International Conference on Very Large Data Bases, August 20-23, 2002, Hong Kong, China*, pp. 1103–1106.

Schadd, F. C. and N. Roos (2011). Improving ontology matchers utilizing linguistic ontologies: an information retrieval approach. In *Proceedings of the 23rd Belgian-Dutch Conference on Artificial Intelligence (BNAIC 2011)*.

Scharffe, F., Y. Liu, and C. Zhou (2009). Rdf-ai: an architecture for rdf datasets matching, fusion and interlink. In *Proc. IJCAI 2009 workshop on Identity, reference, and knowledge representation (IR-KR), Pasadena (CA US)*.

Schmidt, M., O. Görlitz, P. Haase, G. Ladwig, A. Schwarte, and T. Tran (2011). Fedbench: A benchmark suite for federated semantic data query processing. In *The Semantic Web - ISWC 2011 - 10th International Semantic Web Conference, Bonn, Germany, October 23-27, 2011, Proceedings, Part I*, pp. 585–600.

Schmidt, M., T. Hornung, G. Lausen, and C. Pinkel (2009). Spˆ2bench: A SPARQL performance benchmark. In *Proceedings of the 25th International Conference on Data Engineering, ICDE 2009, March 29 2009 - April 2 2009, Shanghai, China*, pp. 222–233.

Schmidt, M., T. Hornung, M. Meier, C. Pinkel, and G. Lausen (2009). SP^2bench: A SPARQL performance benchmark. In *Semantic Web Information Management - A Model-Based Perspective*, pp. 371–393.

Schwarte, A., P. Haase, K. Hose, R. Schenkel, and M. Schmidt (2011). Fedx: Optimization techniques for federated query processing on linked data. In *The Semantic Web - ISWC 2011 - 10th International Semantic Web Conference, Bonn, Germany, October 23-27, 2011, Proceedings, Part I*, pp. 601–616.

Selinger, P. G., M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price (1979). Access path selection in a relational database management system. In *Proceedings of the 1979 ACM SIGMOD international conference on Management of data*, SIGMOD '79, New York, NY, USA, pp. 23–34. ACM.

Settles, B. (2009). Active learning literature survey. Technical Report 1648, University of Wisconsin-Madison.

Settles, B. (2012). *Active Learning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers.

Shekarpour, S., S. Auer, A.-C. Ngonga Ngomo, D. Gerber, S. Hellmann, and C. Stadler (2011). Keyword-driven SPARQL query generation leveraging background knowledge. In *Proceedings of the 2011 IEEE/WIC/ACM International Conference on Web Intelligence, WI 2011, Campus Scientifique de la Doua, Lyon, France, August 22-27, 2011*, pp. 203–210.

Shekarpour, S., A.-C. Ngonga Ngomo, and S. Auer (2013). Question answering on interlinked data. In *22nd International World Wide Web Conference, WWW '13, Rio de Janeiro, Brazil, May 13-17, 2013*, pp. 1145–1156.

Sheridan, J. and J. Tennison (2010). Linking UK government data. In *Proceedings of the WWW2010 Workshop on Linked Data on the Web, LDOW 2010, Raleigh, USA, April 27, 2010*.

Sherif, M. A. and A.-C. Ngonga Ngomo (2015). Semantic quran. *Semantic Web 6*(4), 339–345.

Sherif, M. A., A.-C. Ngonga Ngomo, and J. Lehmann (2015). Automating RDF dataset transformation and enrichment. In *The Semantic Web. Latest Advances and New Domains - 12th European Semantic Web Conference, ESWC 2015, Portoroz, Slovenia, May 31 - June 4, 2015. Proceedings*, pp. 371–387.

Shvaiko, P. and J. Euzenat (2013). Ontology matching: State of the art and future challenges. *IEEE Trans. Knowl. Data Eng. 25*(1), 158–176.

Siegmund, K. D. (2011). Statistical approaches for the analysis of dna methylation microarray data. *Human genetics 129*(6), 585–595.

Silberschatz, A., H. Korth, and S. Sudarshan (2006). *Database Systems Concepts* (5 ed.). New York, NY, USA: McGraw-Hill, Inc.

Sleeman, J. and T. Finin (2010). Computing foaf co-reference relations with rules and machine learning. In *Proceedings of the Third International Workshop on Social Data on the Web*.

Smith, A. and D. Page (2015). U.s. smartphone use in 2015.

Somers, H. (1997). Machine translation and minority languages. *Translating and the Computer*, 13–13.

Song, D. and J. Heflin (2011). Automatically generating data linkages using a domain-independent candidate selection approach. In *The Semantic Web - ISWC 2011 - 10th International Semantic Web Conference, Bonn, Germany, October 23-27, 2011, Proceedings, Part I*, pp. 649–664.

Soru, T., E. Marx, and A.-C. Ngonga Ngomo (2015a). Enhancing dataset quality using keys. In *The Semantic Web - ISWC 2015 - 14th International Semantic Web Conference, Bethelehem, Pennsylvania, USA, October 11-15, 2015, Proceedings, Part II*.

Soru, T., E. Marx, and A.-C. Ngonga Ngomo (2015b). ROCKER: A refinement operator for key discovery. In *Proceedings of the 24th International Conference on World Wide Web, WWW 2015, Florence, Italy, May 18-22, 2015*, pp. 1025–1033.

Soru, T. and A.-C. Ngonga Ngomo (2012). Active learning of domain-specific distances for link discovery. In *Semantic Technology, Second Joint International Conference, JIST 2012, Nara, Japan, December 2-4, 2012. Proceedings*, pp. 97–112.

Soru, T. and A.-C. Ngonga Ngomo (2013). Rapid execution of weighted edit distances. In *Proceedings of the 8th International Workshop on Ontology Matching co-located with the 12th International Semantic Web Conference (ISWC 2013), Sydney, Australia, October 21, 2013.*, pp. 1–12.

Soru, T. and A.-C. Ngonga Ngomo (2014). A comparison of supervised learning classifiers for link discovery. In *Proceedings of the 10th International Conference on Semantic Systems, SEMANTICS 2014, Leipzig, Germany, September 4-5, 2014*, pp. 41–44.

Spearman, C. (1904). The proof and measurement of association between two things. *The American journal of psychology 15*, 72–101.

Speck, R. and A.-C. Ngonga Ngomo (2014). Named entity recognition using FOX. In *Proceedings of the ISWC 2014 Posters & Demonstrations Track a track within the 13th International Semantic Web Conference, ISWC 2014, Riva del Garda, Italy, October 21, 2014.*, pp. 85–88.

Stadler, C., J. Lehmann, K. Höffner, and S. Auer (2012a). Linkedgeodata: A core for a web of spatial open data. *Semantic Web 3*(4), 333–354.

Stadler, C., J. Lehmann, K. Höffner, and S. Auer (2012b). Linkedgeodata: A core for a web of spatial open data. *Semantic Web 3*(4), 333–354.

Stadler, C., M. Martin, J. Lehmann, and S. Hellmann (2010). Update Strategies for DBpedia Live. In *6th Workshop on Scripting and Development for the Semantic Web Colocated with ESWC 2010 30th or 31st May, 2010 Crete, Greece*.

Stadler, C., J. Unbehauen, P. Westphal, M. A. Sherif, and J. Lehmann (2015). Simplified RDB2RDF mapping. In *Proceedings of the 8th Workshop on Linked Data on the Web (LDOW2015), Florence, Italy*.

Stern, R. and B. Sagot (2012). Population of a knowledge base for news metadata from unstructured text and web data. In *Proceedings of the Joint Workshop on Automatic Knowledge Base Construction and Web-scale Knowledge Extraction*, AKBC-WEKEX '12, Stroudsburg, PA, USA, pp. 35–40. Association for Computational Linguistics.

Suchanek, F. M., S. Abiteboul, and P. Senellart (2011). PARIS: probabilistic alignment of relations, instances, and schema. *PVLDB 5*(3), 157–168.

Sutskever, I., R. Salakhutdinov, and J. B. Tenenbaum (2009). Modelling relational data using bayesian clustered tensor factorization. In *Advances in Neural Information Processing Systems 22: 23rd Annual Conference on Neural Information Processing Systems 2009. Proceedings of a meeting held 7-10 December 2009, Vancouver, British Columbia, Canada.*, pp. 1821–1828.

Symeonidou, D., V. Armant, N. Pernelle, and F. Saïs (2014). Sakey: Scalable almost key discovery in RDF data. In *The Semantic Web - ISWC 2014 - 13th International Semantic Web Conference, Riva del Garda, Italy, October 19-23, 2014. Proceedings, Part I*, pp. 33–49.

Tang, M., M. Lee, and Y. J. Kim (2009, July). Interactive hausdorff distance computation for general polygonal models. *ACM Trans. Graph. 28*(3), 74:1–74:9.

Tennison, J., R. Cyganiak, and D. Reynolds (2012, 8). The rdf data cube vocabulary. Technical report, W3C. http://www.w3.org/TR/vocab-data-cube/.

Unger, C., L. Bühmann, J. Lehmann, A.-C. Ngonga Ngomo, D. Gerber, and P. Cimiano (2012). Template-based question answering over RDF data. In *Proceedings of the 21st World Wide Web Conference 2012, WWW 2012, Lyon, France, April 16-20, 2012*, pp. 639–648.

Unger, C. and P. Cimiano (2011). Pythia: Compositional meaning construction for ontology-based question answering on the semantic web. In *Natural Language Processing and Information Systems - 16th International Conference on Applications of Natural Language to Information Systems, NLDB 2011, Alicante, Spain, June 28-30, 2011. Proceedings*, pp. 153–160.

Unger, C., C. Forascu, V. Lopez, A.-C. Ngonga Ngomo, E. Cabrio, P. Cimiano, and S. Walter (2014). Question answering over linked data (QALD-4). In *Working Notes for CLEF 2014 Conference, Sheffield, UK, September 15-18, 2014.*, pp. 1172–1180.

Urbani, J., S. Kotoulas, J. Maassen, F. van Harmelen, and H. E. Bal (2010). OWL reasoning with webpie: Calculating the closure of 100 billion triples. In *The Semantic Web: Research and Applications, 7th Extended Semantic Web Conference, ESWC 2010, Heraklion, Crete, Greece, May 30 - June 3, 2010, Proceedings, Part I*, pp. 213–227.

van Hage, W. R. (2008). *Evaluating Ontology-Alignment Techniques*. Ph. D. thesis, Vrije Universiteit Amsterdam.

Vaquero, L. M., L. Rodero-Merino, and R. Buyya (2011). Dynamically scaling applications in the cloud. *Computer Communication Review 41*(1), 45–52.

Vernica, R., M. J. Carey, and C. Li (2010). Efficient parallel set-similarity joins using mapreduce. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2010, Indianapolis, Indiana, USA, June 6-10, 2010*, pp. 495–506.

Volz, J., C. Bizer, M. Gaedke, and G. Kobilarov (2009). Discovering and maintaining links on the web of data. In *Proceedings of the 8th International Semantic Web Conference*, ISWC '09, Berlin, Heidelberg, pp. 650–665. Springer-Verlag.

Wang, C., J. Wang, X. Lin, W. Wang, H. Wang, H. Li, W. Tian, J. Xu, and R. Li (2010). Mapdupreducer: detecting near duplicates over massive datasets. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2010, Indianapolis, Indiana, USA, June 6-10, 2010*, pp. 1119–1122.

Wang, J., G. Li, and J. Feng (2010). Trie-join: Efficient trie-based string similarity joins with edit-distance constraints. *PVLDB 3*(1), 1219–1230.

Winkler, W. (1999). The state of record linkage and current research problems. Technical report, Statistical Research Division, U.S. Bureau of the Census.

Winkler, W. (2006). Overview of record linkage and current research directions. Technical report, Bureau of the Census - Research Report Series.

Wong, R. C., Y. Tao, A. W. Fu, and X. Xiao (2007). On efficient spatial matching. In *Proceedings of the 33rd International Conference on Very Large Data Bases, University of Vienna, Austria, September 23-27, 2007*, pp. 579–590.

Wu, Z. and M. S. Palmer (1994). Verb semantics and lexical selection. In *32nd Annual Meeting of the Association for Computational Linguistics, 27-30 June 1994, New Mexico State University, Las Cruces, New Mexico, USA, Proceedings.*, pp. 133–138.

Xiao, C., W. Wang, and X. Lin (2008). Ed-Join: an efficient algorithm for similarity joins with edit distance constraints. *PVLDB 1*(1), 933–944.

Xiao, C., W. Wang, X. Lin, and J. X. Yu (2008). Efficient similarity joins for near duplicate detection. In *Proceedings of the 17th International Conference on World Wide Web, WWW 2008, Beijing, China, April 21-25, 2008*, pp. 131–140.

Yao, Z., L. Gao, and X. S. Wang (2003). Using triangle inequality to efficiently process continuous queries on high-dimensional streaming time series. In *Proceedings of the 15th International Conference on Scientific and Statistical Database Management (SSDBM 2003), 9-11 July 2003, Cambridge, MA, USA*, pp. 233–236.

Yosef, M. A., J. Hoffart, I. Bordino, M. Spaniol, and G. Weikum (2011). AIDA: an online tool for accurate disambiguation of named entities in text and tables. *PVLDB 4*(12), 1450–1453.

Yuan, Y. and M. J. Shaw (1995, January). Induction of fuzzy decision trees. *Fuzzy Sets Syst. 69*, 125–139.

Zhai, Y. and B. Liu (2006). Structured data extraction from the web based on partial tree alignment. *IEEE Trans. Knowl. Data Eng. 18*(12), 1614–1628.

COLOPHON

This document was typeset using the typographical look-and-feel `classicthesis` developed by André Miede. The style was inspired by Robert Bringhurst's seminal book on typography "*The Elements of Typographic Style*". `classicthesis` is available for both LaTeX and LyX:

http://code.google.com/p/classicthesis/

Happy users of `classicthesis` usually send a real postcard to the author, a collection of postcards received so far is featured here:

http://postcards.miede.de/

Final Version as of 21. November 2016 (`classicthesis` version 1.0).

DECLARATION

Hiermit erkläre ich, die vorliegende Habilitationsschrift selbständig und ohne unzulässige fremde Hilfe angefertigt zu haben. Ich habe keine anderen als die angeführten Quellen und Hilfsmittel benutzt und sämtliche Textestellen, die wörtlich oder sinngemäß aus veröffentlichten oder unveröffentlichten Schriften entnommen wurden, und alle Angaben, die auf mündlichen Auskünften beruhen, als solche kenntlich gemacht. Ebenfalls sind alle von anderen Personen bereitgestellten Materialien oder erbrachten Dienstleistungen als solche gekennzeichnet.

Leipzig, den 21. November 2016

Axel-Cyrille Ngonga Ngomo